Notes on Randomized Algorithms

James Aspnes

Copyright © 2009–2024 by James Aspnes. Distributed under a Creative Commons Attribution-ShareAlike 4.0 International license: https://creativecommons.org/licenses/by-sa/4.0/.

Contents

Table of contents ii

List of figures xv

List of tables xvi

List of algorithms xvii

Preface xviii

1 Randomized algorithms 1
- 1.1 Searching an array . . . 2
- 1.2 Verifying polynomial identities . . . 4
- 1.3 Randomized QuickSort . . . 6
  - 1.3.1 Brute force method: solve the recurrence . . . 6
  - 1.3.2 Clever method: use linearity of expectation . . . 7
- 1.4 Where does the randomness come from? . . . 9
- 1.5 Classifying randomized algorithms . . . 10
  - 1.5.1 Las Vegas vs Monte Carlo . . . 10
  - 1.5.2 Randomized complexity classes . . . 11
- 1.6 Classifying randomized algorithms by their methods . . . 13

2 Probability theory 15
- 2.1 Probability spaces and events . . . 16
  - 2.1.1 General probability spaces . . . 16
- 2.2 Boolean combinations of events . . . 18
- 2.3 Conditional probability . . . 21
  - 2.3.1 Conditional probability and independence . . . 21
  - 2.3.2 Conditional probability and the law of total probability 21
  - 2.3.3 Examples . . . 23

2.3.3.1 Racing coin-flips . . . 23
2.3.3.2 Karger's min-cut algorithm . . . 25

3 Random variables 28
3.1 Operations on random variables . . . 29
3.2 Random variables and events . . . 29
3.3 Measurability . . . 31
3.4 Expectation . . . 31
3.4.1 Linearity of expectation . . . 32
3.4.1.1 Linearity of expectation for infinite sequences 33
3.4.2 Expectation and inequalities . . . 34
3.4.3 Expectation of a product . . . 35
3.4.3.1 Wald's equation (simple version) . . . 35
3.5 Conditional expectation . . . 36
3.5.1 Expectation conditioned on an event . . . 37
3.5.2 Expectation conditioned on a random variable . . . 38
3.5.2.1 Calculating conditional expectations . . . 39
3.5.2.2 The law of iterated expectation . . . 41
3.5.2.3 Conditional expectation as orthogonal projection . . . 41
3.5.3 Expectation conditioned on a σ-algebra . . . 43
3.5.4 Examples . . . 44
3.6 Applications . . . 45
3.6.1 Yao's lemma . . . 45
3.6.2 Geometric random variables . . . 47
3.6.3 Coupon collector . . . 48
3.6.4 Hoare's FIND . . . 49

4 Basic probabilistic inequalities 51
4.1 Markov's inequality . . . 51
4.1.1 Applications . . . 52
4.1.1.1 Sum of fair coins . . . 52
4.1.1.2 Randomized QuickSort . . . 52
4.1.1.3 Balls in bins . . . 52
4.2 Union bound (Boole's inequality) . . . 53
4.2.1 Example: Balls in bins . . . 53
4.3 Jensen's inequality . . . 54
4.3.1 Proof . . . 55
4.3.2 Applications . . . 56
4.3.2.1 Fair coins: lower bound . . . 56

4.3.2.2 Fair coins: upper bound . . . 56
4.3.2.3 Sifters . . . 57

5 Concentration bounds 59
5.1 Chebyshev's inequality . . . 60
5.1.1 Computing variance . . . 60
5.1.1.1 Alternative formula . . . 60
5.1.1.2 Variance of a Bernoulli random variable . . . 61
5.1.1.3 Variance of a sum . . . 61
5.1.1.4 Variance of a geometric random variable . . . 63
5.1.2 More examples . . . 66
5.1.2.1 Flipping coins . . . 66
5.1.2.2 Balls in bins . . . 66
5.1.2.3 Lazy select . . . 67
5.2 Chernoff bounds . . . 68
5.2.1 The classic Chernoff bound . . . 69
5.2.2 Easier variants . . . 71
5.2.3 Lower bound version . . . 72
5.2.4 Two-sided version . . . 72
5.2.5 What if we only have a bound on E[S]? . . . 73
5.2.6 Almost-independent variables . . . 74
5.2.7 Other tail bounds for the binomial distribution . . . 75
5.2.8 Applications . . . 75
5.2.8.1 Flipping coins . . . 75
5.2.8.2 Balls in bins again . . . 76
5.2.8.3 Flipping coins, central behavior . . . 76
5.2.8.4 Permutation routing on a hypercube . . . 77
5.3 The Azuma-Hoeffding inequality . . . 80
5.3.1 Hoeffding's inequality . . . 80
5.3.1.1 Hoeffding vs Chernoff . . . 82
5.3.1.2 Asymmetric version . . . 83
5.3.2 Azuma's inequality . . . 83
5.3.3 The method of bounded differences . . . 88
5.3.4 Applications . . . 90
5.3.4.1 Sprinkling points on a hypercube . . . 90
5.3.4.2 Chromatic number of a random graph . . . 91
5.3.4.3 Balls in bins . . . 92
5.3.4.4 Probabilistic recurrence relations . . . 92
5.3.4.5 Multi-armed bandits . . . 94
The UCB1 algorithm . . . 95

Analysis of UCB1 . . . 96
5.4 Relation to limit theorems . . . 99
5.5 Anti-concentration bounds . . . 100
5.5.1 The Berry-Esseen theorem . . . 100
5.5.2 The Littlewood-Offord problem . . . 101

6 Randomized search trees **102**
6.1 Binary search trees . . . 102
6.1.1 Rebalancing and rotations . . . 103
6.2 Random insertions . . . 103
6.3 Treaps . . . 105
6.3.1 Assumption of an oblivious adversary . . . 107
6.3.2 Analysis . . . 107
6.3.2.1 Searches . . . 109
6.3.2.2 Insertions and deletions . . . 109
6.3.2.3 Other operations . . . 111
6.4 Skip lists . . . 111

7 Hashing **114**
7.1 Hash tables . . . 114
7.2 Universal hash families . . . 115
7.2.1 Linear congruential hashing . . . 118
7.2.2 Tabulation hashing . . . 119
7.3 FKS hashing . . . 120
7.4 Cuckoo hashing . . . 122
7.4.1 Structure . . . 123
7.4.2 Analysis . . . 124
7.5 Practical issues . . . 127
7.6 Bloom filters . . . 128
7.6.1 Construction . . . 128
7.6.2 False positives . . . 128
7.6.3 Comparison to optimal space . . . 130
7.6.4 Applications . . . 132
7.6.5 Counting Bloom filters . . . 132
7.7 Data stream computation . . . 133
7.7.1 Cardinality estimation . . . 134
7.7.2 Count-min sketches . . . 136
7.7.2.1 Initialization and updates . . . 137
7.7.2.2 Queries . . . 137
7.7.2.3 Finding heavy hitters . . . 140

7.8 Locality-sensitive hashing . . . 140
7.8.1 Approximate nearest neighbor search . . . 140
7.8.2 Locality-sensitive hash functions . . . 141
7.8.3 Constructing an (r_1, r_2)-PLEB . . . 142
7.8.4 Hash functions for Hamming distance . . . 143
7.8.5 Hash functions for ℓ_1 distance . . . 146

8 Dimension reduction **147**
8.1 The Johnson-Lindenstrauss lemma . . . 147
8.1.1 Reduction to single-vector case . . . 148
8.1.2 A relatively simple proof of the lemma . . . 148
8.1.3 Distributional version . . . 151
8.2 Applications . . . 151

9 Martingales and stopping times **153**
9.1 Definitions . . . 153
9.2 Submartingales and supermartingales . . . 154
9.3 The optional stopping theorem . . . 155
9.4 Applications . . . 158
9.4.1 Random walks . . . 158
9.4.2 Wald's equation . . . 162
9.4.3 Maximal inequalities . . . 162
9.4.4 Waiting times for patterns . . . 164

10 Markov chains **165**
10.1 Basic definitions and properties . . . 166
10.1.1 Examples . . . 168
10.2 Convergence of Markov chains . . . 170
10.2.1 Stationary distributions . . . 171
10.2.2 Total variation distance . . . 171
10.2.2.1 Total variation distance and expectation . . . 173
10.2.3 Mixing time . . . 173
10.2.4 Coupling of Markov chains . . . 174
10.2.5 Irreducible and aperiodic chains . . . 175
10.2.6 Convergence of finite irreducible aperiodic Markov chains 177
10.3 Reversible chains . . . 178
10.3.1 Stationary distributions . . . 178
10.3.2 Examples . . . 180
10.3.3 Time-reversed chains . . . 180

10.3.4 Adjusting stationary distributions with the Metropolis-Hastings algorithm . . . 182
10.4 The coupling method . . . 183
10.4.1 Random walk on a cycle . . . 183
10.4.2 Random walk on a hypercube . . . 185
10.4.3 Various shuffling algorithms . . . 186
10.4.3.1 Move-to-top . . . 186
10.4.3.2 Random exchange of arbitrary cards . . . 187
10.4.3.3 Random exchange of adjacent cards . . . 187
10.4.3.4 Real-world shuffling . . . 189
10.4.4 Path coupling . . . 189
10.4.4.1 Random walk on a hypercube . . . 190
10.4.4.2 Sampling graph colorings . . . 191
10.4.4.3 Sampling independent sets . . . 193
10.4.4.4 Simulated annealing . . . 196
Single peak . . . 197
Somewhat smooth functions . . . 197
10.5 Spectral methods for reversible chains . . . 198
10.5.1 Spectral properties of a reversible chain . . . 198
10.5.2 Analysis of symmetric chains . . . 199
10.5.3 Analysis of asymmetric chains . . . 202
10.6 Conductance . . . 203
10.6.1 Easy cases for conductance . . . 204
10.6.2 Edge expansion using canonical paths . . . 204
10.6.3 Congestion . . . 206
10.6.4 Examples . . . 208
10.6.4.1 Lazy random walk on a line . . . 208
10.6.4.2 Random walk on a hypercube . . . 208
10.6.4.3 Matchings in a graph . . . 209
10.6.4.4 Perfect matchings in dense bipartite graphs . 211

11 Approximate counting **214**
11.1 Exact counting . . . 214
11.2 Counting by sampling . . . 215
11.2.1 Generating samples . . . 216
11.3 Approximating #KNAPSACK . . . 217
11.4 Approximating #DNF . . . 220
11.5 Approximating exponentially improbable events . . . 221
11.5.1 Matchings . . . 222
11.5.2 Other problems . . . 223

12 Hitting times **224**
12.1 Waiting times . . . 224
12.2 Lyapunov functions . . . 225
12.3 Drift analysis . . . 230

13 The probabilistic method **234**
13.1 Randomized constructions and existence proofs . . . 234
13.1.1 Set balancing . . . 235
13.1.2 Ramsey numbers . . . 235
13.2 Approximation algorithms . . . 237
13.2.1 MAX CUT . . . 238
13.2.2 MAX SAT . . . 239
13.3 The Lovász Local Lemma . . . 242
13.3.1 General version . . . 243
13.3.2 Symmetric version . . . 244
13.3.3 Applications . . . 244
13.3.3.1 Graph coloring . . . 244
13.3.3.2 Satisfiability of k-CNF formulas . . . 245
13.3.3.3 Hypergraph 2-colorability . . . 245
13.3.4 Non-constructive proof . . . 246
13.3.5 Constructive proof . . . 249

14 Derandomization **253**
14.1 Deterministic vs. randomized algorithms . . . 254
14.2 Adleman's theorem . . . 255
14.3 Limited independence . . . 256
14.3.1 MAX CUT . . . 256
14.4 The method of conditional probabilities . . . 257
14.4.1 MAX CUT using conditional probabilities . . . 258
14.4.2 Deterministic construction of Ramsey graphs . . . 259
14.4.3 Derandomized set balancing . . . 259

15 Probabilistically-checkable proofs **261**
15.1 Probabilistically-checkable proofs . . . 262
15.2 A PCP for GRAPH NON-ISOMORPHISM . . . 263
15.2.1 GRAPH NON-ISOMORPHISM with private coins . . . 263
15.2.2 A probabilistically-checkable proof for GRAPH NON-ISOMORPHISM . . . 264
15.3 $\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), 1)$ . . . 264
15.3.1 QUADEQ . . . 264

15.3.2 The Walsh-Hadamard Code . . . 265
15.3.3 A PCP for QUADEQ . . . 266
15.4 **PCP** and approximability . . . 267
15.4.1 Approximating the number of satisfied verifier queries 267
15.4.2 Gap-preserving reduction to MAX SAT . . . 268
15.4.3 Other inapproximable problems . . . 269
15.5 Dinur's proof of the **PCP** theorem . . . 270
15.6 The Unique Games Conjecture . . . 272

16 Quantum computing **274**
16.1 Random circuits . . . 274
16.2 Bra-ket notation . . . 277
16.2.1 States as kets . . . 277
16.2.2 Composition of kets . . . 278
16.2.3 Operators as sums of kets times bras . . . 278
16.3 Quantum circuits . . . 279
16.3.1 Quantum operations . . . 280
16.3.2 Quantum implementations of classical operations . . . 281
16.3.3 Phase representation of a function . . . 283
16.3.4 Practical issues (which we will ignore) . . . 284
16.3.5 Quantum computations . . . 284
16.4 Deutsch's algorithm . . . 284
16.5 Grover's algorithm . . . 285
16.5.1 Initial superposition . . . 286
16.5.2 The Grover diffusion operator . . . 286
16.5.3 Effect of the iteration . . . 287

17 Randomized distributed algorithms **289**
17.1 Consensus . . . 290
17.1.1 Impossibility of deterministic algorithms . . . 291
17.2 Leader election . . . 292
17.3 How randomness helps . . . 292
17.4 Building a weak shared coin . . . 293
17.5 Leader election with sifters . . . 295
17.6 Consensus with sifters . . . 296

A Sample assignments from Spring 2024 **299**
A.1 Assignment 1, due Thursday 2024-02-01 at 23:59 . . . 299
A.1.1 Matchings . . . 299
A.1.2 Non-volatile memory . . . 301

A.2 Assignment 2, due Thursday 2024-02-15 at 23:59 304
A.2.1 Mediocre cuts 304
A.2.2 Training costs 305
A.3 Assignment 3, due Thursday 2024-02-29 at 23:59 307
A.3.1 A robot rendezvous problem 307
A.3.2 A linked list 308
A.4 Assignment 4, due Thursday 2024-03-28 at 23:59 311
A.4.1 Nearly orthogonal vectors 311
A.4.2 Boosting a random walk 312
A.5 Assignment 5, due Thursday 2024-04-11 at 23:59 315
A.5.1 Relaxation time for Metropolis-Hastings 315
A.5.2 A constrained random walk 317
A.6 Assignment 6, due Thursday 2024-04-25 at 23:59 319
A.6.1 The power of intransigence 319
Potential function argument 321
Coupling argument 321
A.6.2 Almost Markov 322

B Sample assignments from Spring 2023 **324**
B.1 Assignment 1, due Thursday 2023-02-16 at 23:59 324
B.1.1 Hashing without counting 324
B.1.2 Permutation routing on an incomplete network 326
B.2 Assignment 2, due Thursday 2023-03-30 at 23:59 331
B.2.1 Some streaming data structures 331
B.2.2 A dense network 333
B.3 Assignment 3, due Thursday 2023-04-27 at 23:59 335
B.3.1 Shuffling a graph 335
B.3.2 Counting unbalanced sets 337

C Sample assignments from Fall 2019 **340**
C.1 Assignment 1: due Thursday, 2019-09-12, at 23:00 340
C.1.1 The golden ticket 340
C.1.2 Exploding computers 342
C.2 Assignment 2: due Thursday, 2019-09-26, at 23:00 344
C.2.1 A logging problem 344
C.2.2 Return of the exploding computers 345
C.3 Assignment 3: due Thursday, 2019-10-10, at 23:00 346
C.3.1 Two data plans 346
C.3.2 A randomly-indexed list 348
C.4 Assignment 4: due Thursday, 2019-10-31, at 23:00 349

C.4.1 A hash tree . . . 349
C.4.2 Randomized robot rendezvous on a ring . . . 350
C.5 Assignment 5: due Thursday, 2019-11-14, at 23:00 . . . 352
C.5.1 Non-exploding computers . . . 352
C.5.2 A wordy walk . . . 354
C.6 Assignment 6: due Monday, 2019-12-09, at 23:00 . . . 356
C.6.1 Randomized colorings . . . 356
C.6.2 No long paths . . . 357
C.7 Final exam . . . 359
C.7.1 A uniform ring . . . 359
C.7.2 Forbidden runs . . . 361
C.7.3 A derandomized balancing scheme . . . 362

D Sample assignments from Fall 2016 364
D.1 Assignment 1: due Sunday, 2016-09-18, at 17:00 . . . 364
D.1.1 Bubble sort . . . 364
D.1.2 Finding seats . . . 366
D.2 Assignment 2: due Thursday, 2016-09-29, at 23:00 . . . 368
D.2.1 Technical analysis . . . 368
D.2.2 Faulty comparisons . . . 372
D.3 Assignment 3: due Thursday, 2016-10-13, at 23:00 . . . 373
D.3.1 Painting with sprites . . . 373
D.3.2 Dynamic load balancing . . . 375
D.4 Assignment 4: due Thursday, 2016-11-03, at 23:00 . . . 376
D.4.1 Re-rolling a random treap . . . 376
D.4.2 A defective hash table . . . 379
D.5 Assignment 5: due Thursday, 2016-11-17, at 23:00 . . . 381
D.5.1 A spectre is haunting Halloween . . . 381
D.5.2 Colliding robots on a line . . . 382
D.6 Assignment 6: due Thursday, 2016-12-08, at 23:00 . . . 386
D.6.1 Another colliding robot . . . 386
D.6.2 Return of the sprites . . . 389
D.7 Final exam . . . 391
D.7.1 Virus eradication (20 points) . . . 391
D.7.2 Parallel bubblesort (20 points) . . . 392
D.7.3 Rolling a die (20 points) . . . 393

E Sample assignments from Spring 2014 **395**
E.1 Assignment 1: due Wednesday, 2014-09-10, at 17:00 395
E.1.1 Bureaucratic part 395
E.1.2 Two terrible data structures 395
E.1.3 Parallel graph coloring 399
E.2 Assignment 2: due Wednesday, 2014-09-24, at 17:00 401
E.2.1 Load balancing 401
E.2.2 A missing hash function 401
E.3 Assignment 3: due Wednesday, 2014-10-08, at 17:00 402
E.3.1 Tree contraction 402
E.3.2 Part testing 405
Using McDiarmid's inequality and some cleverness 406
E.4 Assignment 4: due Wednesday, 2014-10-29, at 17:00 407
E.4.1 A doubling strategy 407
E.4.2 Hash treaps 408
E.5 Assignment 5: due Wednesday, 2014-11-12, at 17:00 409
E.5.1 Agreement on a ring 409
E.5.2 Shuffling a two-dimensional array 412
E.6 Assignment 6: due Wednesday, 2014-12-03, at 17:00 413
E.6.1 Sampling colorings on a cycle 413
E.6.2 A hedging problem 414
E.7 Final exam 415
E.7.1 Double records (20 points) 416
E.7.2 Hipster graphs (20 points) 417
Using the method of conditional probabilities . 417
Using hill climbing 418
E.7.3 Storage allocation (20 points) 418
E.7.4 Fault detectors in a grid (20 points) 419

F Sample assignments from Spring 2013 **422**
F.1 Assignment 1: due Wednesday, 2013-01-30, at 17:00 422
F.1.1 Bureaucratic part 422
F.1.2 Balls in bins 422
F.1.3 A labeled graph 423
F.1.4 Negative progress 423
F.2 Assignment 2: due Thursday, 2013-02-14, at 17:00 425
F.2.1 A local load-balancing algorithm 425
F.2.2 An assignment problem 427
F.2.3 Detecting excessive collusion 427
F.3 Assignment 3: due Wednesday, 2013-02-27, at 17:00 429

F.3.1 Going bowling . . . 429
F.3.2 Unbalanced treaps . . . 430
F.3.3 Random radix trees . . . 432
F.4 Assignment 4: due Wednesday, 2013-03-27, at 17:00 . . . 433
F.4.1 Flajolet-Martin sketches with deletion . . . 433
F.4.2 An adaptive hash table . . . 435
F.4.3 An odd locality-sensitive hash function . . . 437
F.5 Assignment 5: due Friday, 2013-04-12, at 17:00 . . . 438
F.5.1 Choosing a random direction . . . 438
F.5.2 Random walk on a tree . . . 439
F.5.3 Sampling from a tree . . . 440
F.6 Assignment 6: due Friday, 2013-04-26, at 17:00 . . . 441
F.6.1 Increasing subsequences . . . 441
F.6.2 Futile word searches . . . 442
F.6.3 Balance of power . . . 444
F.7 Final exam . . . 445
F.7.1 Dominating sets . . . 445
F.7.2 Tricolor triangles . . . 446
F.7.3 The n rooks problem . . . 447
F.7.4 Pursuing an invisible target on a ring . . . 447

G Sample assignments from Spring 2011 **449**
G.1 Assignment 1: due Wednesday, 2011-01-26, at 17:00 . . . 449
G.1.1 Bureaucratic part . . . 449
G.1.2 Rolling a die . . . 449
G.1.3 Rolling many dice . . . 451
G.1.4 All must have candy . . . 451
G.2 Assignment 2: due Wednesday, 2011-02-09, at 17:00 . . . 452
G.2.1 Randomized dominating set . . . 452
G.2.2 Chernoff bounds with variable probabilities . . . 454
G.2.3 Long runs . . . 455
G.3 Assignment 3: due Wednesday, 2011-02-23, at 17:00 . . . 457
G.3.1 Longest common subsequence . . . 457
G.3.2 A strange error-correcting code . . . 459
G.3.3 A multiway cut . . . 460
G.4 Assignment 4: due Wednesday, 2011-03-23, at 17:00 . . . 461
G.4.1 Sometimes successful betting strategies are possible . . . 461
G.4.2 Random walk with reset . . . 463
G.4.3 Yet another shuffling algorithm . . . 465
G.5 Assignment 5: due Thursday, 2011-04-07, at 23:59 . . . 466

G.5.1 A reversible chain . . . 466
G.5.2 Toggling bits . . . 467
G.5.3 Spanning trees . . . 469
G.6 Assignment 6: due Monday, 2011-04-25, at 17:00 . . . 470
G.6.1 Sparse satisfying assignments to DNFs . . . 470
G.6.2 Detecting duplicates . . . 471
G.6.3 Balanced Bloom filters . . . 472
G.7 Final exam . . . 475
G.7.1 Leader election . . . 475
G.7.2 Two-coloring an even cycle . . . 476
G.7.3 Finding the maximum . . . 477
G.7.4 Random graph coloring . . . 478

H Sample assignments from Spring 2009 **479**
H.1 Final exam, Spring 2009 . . . 479
H.1.1 Randomized mergesort (20 points) . . . 479
H.1.2 A search problem (20 points) . . . 480
H.1.3 Support your local police (20 points) . . . 481
H.1.4 Overloaded machines (20 points) . . . 482

I Probabilistic recurrences **483**
I.1 Recurrences with constant cost functions . . . 483
I.2 Examples . . . 483
I.3 The Karp-Upfal-Wigderson bound . . . 484
I.3.1 Waiting for heads . . . 486
I.3.2 Quickselect . . . 486
I.3.3 Tossing coins . . . 486
I.3.4 Coupon collector . . . 487
I.3.5 Chutes and ladders . . . 487
I.4 High-probability bounds . . . 488
I.4.1 High-probability bounds from expectation bounds . . . 489
I.4.2 Detailed analysis of the recurrence . . . 489
I.5 More general recurrences . . . 490

Bibliography **491**

Index **510**

List of Figures

2.1 Karger's min-cut algorithm 26

4.1 Proof of Jensen's inequality 56

5.1 Comparison of Chernoff bound variants 72
5.2 Hypercube network with $n = 3$ 77

6.1 Tree rotations 103
6.2 Balanced and unbalanced binary search trees 104
6.3 Binary search tree after inserting 5 1 7 3 4 6 2 105
6.4 Inserting values into a treap 106
6.5 Tree rotation shortens spines 110
6.6 Skip list 111

10.1 Drawing a Markov chain as a digraph 167

13.1 Tricky step in MAX SAT argument 242

A.1 Non-volatile memory in action 302

B.1 An incomplete network with $n = 2d = 4$ 327
B.2 Graph density evolution 333

D.1 Filling a screen with Space Invaders 374
D.2 Hidden Space Invaders 390

E.1 Example of tree contraction for Problem E.3.1 403

F.1 Radix tree 432
F.2 Word searches 443

List of Tables

3.1 Sum of two dice 30
3.2 Various conditional expectations on two independent dice . . 46

5.1 Concentration bounds 60

7.1 Hash table parameters 116

List of Algorithms

7.1 Insertion procedure for cuckoo hashing 124

17.1 Attiya-Censor weak shared coin [AC08] 294
17.2 Giakkoupis-Woelfel sifter [GW12] 295
17.3 Sifter using max registers [AE19] 298

D.1 One pass of bubble sort 365

F.1 Adaptive hash table insertion 436

G.1 Dubious duplicate detector 471
G.2 Randomized max-finding algorithm 477

Preface

These are notes for the Yale course CPSC 469/569 Randomized Algorithms. This document also incorporates the lecture schedule and assignments, as well as some sample assignments from previous semesters. Because this is a work in progress, it will be updated frequently over the course of the semester.

Much of the structure of the course follows Mitzenmacher and Upfals's *Probability and Computing* [MU17], with some material from Motwani and Raghavan's *Randomized Algorithms* [MR95]. In most cases you'll find these textbooks contain much more detail than what is presented here, so it is probably better to consider this document a supplement to them than to treat it as your primary source of information.

The most recent version of these notes will be available at `https://www.cs.yale.edu/homes/aspnes/classes/469/notes.pdf`. More stable archival versions may be found at `https://arxiv.org/abs/2003.01902`.

I would like to thank my many students and teaching fellows over the years for their help in pointing out errors and omissions in earlier drafts of these notes.

Chapter 1

Randomized algorithms

A randomized algorithm flips coins during its execution to determine what to do next. When considering a randomized algorithm, we usually care about its **expected worst-case** performance, which is the average amount of time it takes on the worst input of a given size. This average is computed over all the possible outcomes of the coin flips during the execution of the algorithm. We may also ask for a **high-probability bound**, showing that the algorithm doesn't consume too much resources most of the time.

In studying randomized algorithms, we consider pretty much the same issues as for deterministic algorithms: how to design a good randomized algorithm, and how to prove that it works within given time or error bounds. The main difference is that it is often easier to design a randomized algorithm—randomness turns out to be a good substitute for cleverness more often than one might expect—but harder to analyze it. So much of what one does is develop good techniques for analyzing the often very complex random processes that arise in the execution of an algorithm. Fortunately, in doing so we can often use techniques already developed by probabilists and statisticians for analyzing less overtly algorithmic processes.

Formally, we think of a randomized algorithm as a machine M that computes $M(x, r)$, where x is the problem input and r is the sequence of random bits. Our machine model is the usual **random-access machine** or **RAM** model, where we have a memory space that is typically polynomial in the size of the input n, and in constant time we can read a memory location, write a memory location, or perform arithmetic operations on integers of up to $O(\log n)$ bits.[1] In this model, we may find it easier to think of the

[1] This model is unrealistic in several ways: the assumption that we can perform arithmetic on $O(\log n)$-bit quantities in constant time omits at least a factor of $\Omega(\log \log n)$ for addition

random bits as supplied as needed by some subroutine, where generating a random integer of size $O(\log n)$ takes constant time; the justification for this assumption is that it takes constant time to read the next $O(\log n)$-sized value from the random input.

Because the number of these various constant-time operations, and thus the running time for the algorithm as a whole, may depend on the random bits, it is now a **random variable**—a function on points in some probability space. The probability space Ω consists of all possible sequences r, each of which is assigned a probability $\Pr[r]$ (typically $2^{-|r|}$), and the running time for M on some input x is generally given as an **expected value**[2] $\mathrm{E}_r[\mathrm{time}(M(x,r))]$, where for any X,

$$\mathrm{E}_r[X] = \sum_{r\in\Omega} X(r)\Pr[r]\,. \tag{1.0.1}$$

We can now quote the performance of M in terms of this expected value: where we would say that a deterministic algorithms runs in time $O(f(n))$, where $n = |x|$ is the size of the input, we instead say that our randomized algorithm runs in **expected time** $O(f(n))$, which means that $\mathrm{E}_r[\mathrm{time}(M(x,r))] = O(f(|x|))$ for all inputs x.

This is distinct from traditional **worst-case analysis**, where there is no r and no expectation, and **average-case analysis**, where there is again no r and the value reported is not a maximum but an expectation over some distribution on x. The following trivial example shows the distinction.

1.1 Searching an array

Suppose we have an array A with n elements, and we are told that one of the positions in the array contains a nonzero value (the "prize") while every other position contains zero. How quickly can we find the prize in the worst case?

Any *deterministic* algorithm will probe the array positions in some predictable order. An adversary can simulate the algorithm running in an

and probably more for multiplication in any realistic implementation, while the assumption that we can address n^c distinct locations in memory in anything less than $n^{c/3}$ time in the worst case requires exceeding the speed of light. But for reasonably small n, this gives a pretty good approximation of the performance of real computers, which do in fact perform arithmetic and access memory in a fixed amount of time, although with fixed bounds on the size of both arithmetic operands and memory.

[2] We'll see more details of these and other concepts from probability theory in Chapters 2 and 3.

execution where it always sees 0 until it has checked every location. By putting the prize in the last place the algorithm looks, we get a worst-case input that requires at least n probes to find it.

In the average case, we might assume that instead of placing the prize in the worst possible position, it is placed uniformly at random. Now if we just scan the array from left to right, the expected number of probes will be

$$\begin{aligned}\sum_{i=1}^{n} i \cdot \Pr\left[\text{algorithm does } i \text{ probes}\right] &= \sum_{i=1}^{n} i \cdot \frac{1}{n} \\ &= \frac{1}{n} \cdot \frac{n(n+1)}{2} \\ &= \frac{n+1}{2}.\end{aligned}$$

This is still $\Theta(n)$, but we've improved the constant factor by almost a factor of 2. The cost is that we trust the adversary to place the prize randomly—but there is no reason for the adversary to do this.

The trick to randomized algorithms is that we can can obtain the same expected payoff even in the worst case by supplying the randomness ourselves. Let's suppose that instead of scanning the array predictably from left to right, we flip a coin and either scan left-to-right or scan right-to-left with probability 1/2 each. If the prize is in position i, then scanning left-to-right finds it in i probes, while scanning right-to-left finds it in $n - i + 1$ probes. The expected cost is thus

$$\frac{1}{2} \cdot i + \frac{1}{2} \cdot (n - i + 1) = \frac{n+1}{2},$$

the same as in the average case. But now we don't make any assumptions about the input—by using just one bit of randomness we get this expected time no matter where the adversary puts the prize.

A natural question is whether a more clever algorithm can do even better. Usually it's pretty hard to prove lower bounds on algorithms, but in this case we can use a classic result called **Yao's lemma** [Yao77] to show a worst-case lower bound on the expected cost of any randomized algorithm by constructing a probability distribution on inputs that gives a bad average-case lower bound for any deterministic algorithm. We'll give a proof of this in §3.6.1, but the intuition is that any randomized algorithm acts like picking a deterministic algorithm at random, so if there is an input distribution that is bad for all deterministic algorithms, it is still bad even if we pick one of those algorithms randomly.

In this case, the bad input distribution is simple: put the 1 in each position $A[i]$ with equal probability $1/n$. For a deterministic algorithm, there will be some fixed sequence of positions $i_1, i_2, \ldots$ that it examines as long as it only sees zeros. A smart deterministic algorithm will not examine the same position twice, so the 1 is equally likely to be found $1, 2, 3, \ldots, n$ probes. This gives the same expected $\frac{n+1}{2}$ probes as for the simple randomized algorithm, which shows that that algorithm is optimal.

We've been talking about searching an array, because that fits best in our model of an input supplied to the algorithm, but essentially the same analysis applies to brute-force inverting a black-box function. Here we have a function f and target output y, and we want to find an input x such that $f(x) = y$. The same analysis as for the array case shows that this takes $\frac{n+1}{2}$ expected evaluations of f assuming that exactly one x works and we can't do anything clever.

Curiously, in this case it may be possible to improve this bound to $O(\sqrt{n})$ evaluations if somehow we get our hands on a working quantum computer. We'll come back to this when we discuss quantum computing in general and Grover's algorithm in particular in Chapter 16.

1.2 Verifying polynomial identities

This classic problem is described in [MU17, §1.1]. Here we are given two products of polynomials and we want to determine if they compute the same function. For example, we might have

$$p(x) = (x-7)(x-3)(x-1)(x+2)(2x+5)$$
$$q(x) = 2x^5 - 13x^4 - 21x^3 + 127x^2 + 121x - 210$$

These expressions both represent degree-5 polynomials, and it is not obvious without multiplying out the factors of p whether they are equal or not. Multiplying out all the factors of p may take as much as $O(d^2)$ time if we assume integer multiplication takes unit time and do it the straightforward way.[3] We can do better than this using randomization.

The trick is that evaluating $p(x)$ and $q(x)$ takes only $O(d)$ integer operations, and we will find $p(x) = q(x)$ only if either (a) $p(x)$ and $q(x)$ are really the same polynomial, or (b) x is a **root** of $p(x) - q(x)$. Since $p(x) - q(x)$ has degree at most d, it can't have more than d roots. So if we choose x uniformly at random from some much larger space, it's likely that we will not

[3] It can be faster if we do something sneaky like use fast Fourier transforms [SS71].

get a root. Indeed, evaluating $p(11) = 112320$ and $q(11) = 120306$ quickly shows that p and q are not in fact the same.

This is an example of a **Monte Carlo algorithm**, which is an algorithm that runs in a fixed amount of time but only gives the right answer some of the time. (In this case, with probability $1 - d/r$, where r is the size of the range of random integers we choose x from.) Monte Carlo algorithms have the unnerving property of not indicating when their results are incorrect, but we can make the probability of error as small as we like by running the algorithm repeatedly. For this particular algorithm, the probability of error after k trials is only $(d/r)^k$, which means that for fixed d/r we need $O(\log(1/\epsilon))$ iterations to get the error bound down to any given ϵ. If we are really paranoid, we could get the error down to 0 by testing $d + 1$ distinct values, but now the cost is as high as multiplying out p again.

The error for this algorithm is one-sided: if we find a **witness** to the fact that $p \neq q$, we are done, but if we don't, then all we know is that we haven't found a witness yet. We also have the property that if we check enough possible witnesses, we are guaranteed to find one.

A similar property holds in the classic **Miller-Rabin primality test**, a randomized algorithm for determining whether a large integer is prime or not.[4] The original version, due to Gary Miller [Mil76] showed that, as in polynomial identity testing, it might be sufficient to pick a particular set of deterministic candidate witnesses. Unfortunately, this result depends on the truth of the extended Riemann hypothesis, a notoriously difficult open problem in number theory. Michael Rabin [Rab80] demonstrated that choosing random witnesses was enough, if we were willing to accept a small probability of incorrectly identifying a composite number as prime.

For many years it was open whether it was possible to test primality deterministically in polynomial time without unproven number-theoretic assumptions, and the randomized Miller-Rabin algorithm was one of the most widely-used randomized algorithms for which no good deterministic alternative was known. Eventually, Agrawal *et al.* [AKS04] demonstrated how to test primality deterministically using a different technique, although the cost of their algorithm is high enough that Miller-Rabin is still used in practice.

[4] We will not describe this algorithm here.

1.3 Randomized QuickSort

The **QuickSort** algorithm [Hoa61a] works as follows. For simplicity, we assume that no two elements of the array being sorted are equal.

- If the array has > 1 elements,
  - Pick a **pivot** p uniformly at random from the elements of the array.
  - Split the array into A_1 and A_2, where A_1 contains all elements $< p$ elements $> p$.
  - Sort A_1 and A_2 recursively and return the sequence A_1, p, A_2.
- Otherwise return the array.

The splitting step takes exactly $n-1$ comparisons, since we have to check each non-pivot against the pivot. We assume all other costs are dominated by the cost of comparisons. How many comparisons does randomized QuickSort do on average?

There are two ways to solve this problem: the dumb way and the smart way. We'll do it the dumb way now and save the smart way for §1.3.2.

1.3.1 Brute force method: solve the recurrence

Let $T(n)$ be the expected number of comparisons done on an array of n elements. We have $T(0) = T(1) = 0$ and for larger n,

$$T(n) = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-1-k)\right). \tag{1.3.1}$$

Why? Because we do $(n-1)$ comparisons to split the piles, there are n equally-likely choices for our pivot (hence the $1/n$), and for each choice the expected cost of the recursive sorts is $T(k) + T(n-1-k)$, where k is the number of elements that land in A_1. Formally, we are using here the **law of total probability**, which says that for any random variable X and partition of the probability space into events $B_1 \ldots B_n$, then

$$\mathrm{E}[X] = \sum \Pr[B_i]\,\mathrm{E}[X \mid B_i],$$

where

$$\mathrm{E}[X \mid B_i] = \sum_{\omega \in B_i} X(\omega)\frac{\Pr[\omega]}{\Pr[B_i]}$$

is the **conditional expectation** of X conditioned on B_i, which we can think of as just the average value of X if we know that B_i occurred. (See §2.3.2 for more details.)

So now we just have to solve this ugly recurrence. We can reasonably guess that when $n \geq 1, T(n) \leq an \log n$ for some constant a. Clearly this holds for $n = 1$. Now apply induction on larger n to get

$$\begin{aligned}
T(n) &= (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-1-k)) \\
&= (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} T(k) \\
&= (n-1) + \frac{2}{n}\sum_{k=1}^{n-1} T(k) \\
&\leq (n-1) + \frac{2}{n}\sum_{k=1}^{n-1} ak \log k \\
&\leq (n-1) + \frac{2}{n}\int_{k=1}^{n} ak \log k \\
&= (n-1) + \frac{2a}{n}\left(\frac{n^2 \log n}{2} - \frac{n^2}{4} + \frac{1}{4}\right) \\
&= (n-1) + an \log n - \frac{an}{2} + \frac{a}{2n}.
\end{aligned}$$

If we squint carefully at this recurrence for a while we notice that setting $a = 2$ makes this less than or equal to $an \log n$, since the remaining terms become $(n-1) - n + 1/n = 1/n - 1$, which is negative for $n \geq 1$. We can thus confidently conclude that $T(n) \leq 2n \log n$ (for $n \geq 1$).

1.3.2 Clever method: use linearity of expectation

Alternatively, we can use **linearity of expectation** (which we'll discuss further in §3.4.1) to compute the expected number of comparisons used by randomized QuickSort.

Imagine we use the following method for choosing pivots: we generate a random permutation of all the elements in the array, and when asked to sort some subarray A', we use as pivot the first element of A' that appears in our list. Since each element is equally likely to be first, this is equivalent to the actual algorithm. Pretend that we are always sorting the numbers $1 \ldots n$ and define for each pair of elements $i < j$ the **indicator variable** X_{ij} to be 1 if

i is compared to j at some point during the execution of the algorithm and 0 otherwise. Amazingly, we can actually compute the probability of this event (and thus $\mathrm{E}[X_{ij}]$): the only time i and j are compared is if one of them is chosen as a pivot before they are split up into different arrays. How do they get split up into different arrays? This happens if some intermediate element k is chosen as pivot first, that is, if some k with $i < k < j$ appears in the permutation before both i and j. Occurrences of other elements don't affect the outcome, so we can concentrate on the restriction of the permutations to just the numbers i through j, and we win if this restricted permutation starts with either i or j. This event occurs with probability $2/(j-i+1)$, so we have $\mathrm{E}[X_{ij}] = 2/(j-i+1)$. Summing over all pairs $i < j$ gives:

$$\begin{aligned}
\mathrm{E}\left[\sum_{i<j} X_{ij}\right] &= \sum_{i<j} \mathrm{E}[X_{ij}] \\
&= \sum_{i<j} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \\
&= \sum_{i=2}^{n} \sum_{k=2}^{i} \frac{2}{k} \\
&= \sum_{k=2}^{n} \frac{2(n-k+1)}{k} \\
&= \sum_{k=2}^{n} \left(\frac{2(n+1)}{k} - 2\right) \\
&= \sum_{k=2}^{n} \frac{2(n+1)}{k} - 2(n-1) \\
&= 2(n+1)(H_n - 1) - 2(n-1) \\
&= 2(n+1)H_n - 4n.
\end{aligned}$$

Here $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th **harmonic number**, equal to $\ln n + \gamma + O(n^{-1})$, where $\gamma \approx 0.5772$ is the **Euler-Mascheroni constant** (whose exact value is unknown!). For asymptotic purposes we only need $H_n = \Theta(\log n)$.

For the first step we are taking advantage of the fact that linearity of expectation doesn't care about the variables not being independent. The rest is just algebra.

This is pretty close to the bound of $2n \log n$ we computed using the

recurrence in §1.3.1. Given that we now know the exact answer, we could in principle go back and use it to solve the recurrence exactly.[5]

Which way is better? Solving the recurrence requires less probabilistic **handwaving** (a more polite term might be "insight") but more grinding out inequalities, which is a pretty common trade-off. Since I am personally not very clever I would try the brute-force approach first. But it's worth knowing about better methods so you can try them in other situations.

1.4 Where does the randomness come from?

Typically we assume in analyzing a randomized algorithm that we have access to genuinely random bits r, and don't ask too carefully how we can get these random bits. For practical applications, there are basically three choices, from strongest (and most expensive) to weakest (and cheapest):

1. **Physical randomness.** Lock a cat in a box with a radioactive source and a Geiger counter attached to a solenoid aimed at a vial of prussic acid [Sch35]. Come back in an hour and check if the cat is still breathing. With an appropriately-tuned radioactive source, this generates one very unpredictable bit of randomness at the cost of one half of a cat on average.[6]

   There are cheaper variants, mostly involving amplified quantum noise at the high end, or monitoring physical processes that are expected to be somewhat random (like intervals between keyboard presses or seek times of hard drive heads) at the low end. In each case you get a sequence of random bits that, under plausible physical assumptions, are effectively unpredictable.

   The `/dev/random` device in Linux systems gives you access to the cheap kind of physical randomness, and will block waiting for you to wave your mouse around if it runs out.

2. **Cryptographically-secure pseudorandomness.** Find some function that spits out a sequence of random-looking values given a **seed**, such that if the seed is chosen uniformly at random (say using a physical random number generator), no polynomial-time program can distinguish the sequence from an actual random sequence with non-trivial

[5] We won't.

[6] Expected cost may be higher if the observer forgets their gas mask.

probability. Usually expensive, but if your polynomial-time randomized algorithm fails using a CPRNG, you've succeeded in breaking its cryptographic assumptions.

The `/dev/urandom` device in Linux systems gives you this, based on a seed derived from the same sources as `/dev/random`.

3. **Statistical pseudorandomness.** As above, but choose a function that is not cryptographically secure but merely passes common statistical tests for things like k-wise independence of consecutive outputs. Which function to choose is largely a matter of convenience and fashion (although PRNGs in older standard libraries can be very, very bad). The advantage is that many PRNGs are very fast and will not slow your program down. The disadvantage is that you are relying on your program not doing anything that exposes the weakness of the PRNG.

   The `random` function in the standard C library is an example of this, although maybe not a good example. The cool kids mostly use Mersenne Twister [MN98].

One advantage of pseudorandom generators is that they allow for debugging: if you run your program twice with the same key, it will do the same thing. This was in fact an argument made by von Neumann *against* using physical randomness back in the old days [VN63].

In practice, most people just use whatever PRNG is ready to hand, and hope it works. As theorists, we will ignore this issue, and assume that our random inputs are actually random.

1.5 Classifying randomized algorithms

Different random algorithms make different guarantees about the likelihood of getting a correct output or even the possibility of recognizing a correct output. These different guarantees have established names in the randomized algorithms literature and are correspond to various complexity classes in complexity theory.

1.5.1 Las Vegas vs Monte Carlo

One difference between QuickSort and polynomial equality testing that QuickSort always succeeds, but may run for longer than you expect; while the polynomial equality tester always runs in a fixed amount of time, but may

produce the wrong answer. These are examples of two classes of randomized algorithms, which were originally named by László Babai [Bab79]:[7]

- A **Las Vegas algorithm** fails with some probability, but we can tell when it fails. In particular, we can run it again until it succeeds, which means that we can eventually succeed with probability 1 (but with a potentially unbounded running time). Alternatively, we can think of a Las Vegas algorithm as an algorithm that runs for an unpredictable amount of time but always succeeds (we can convert such an algorithm back into one that runs in bounded time by declaring that it fails if it runs too long—a condition we can detect). QuickSort is an example of a Las Vegas algorithm.
- A **Monte Carlo algorithm** fails with some probability, but we can't tell when it fails. If the algorithm produces a yes/no answer and the failure probability is significantly less than 1/2, we can reduce the probability of failure by running it many times and taking a majority of the answers. The polynomial equality-testing algorithm in an example of a Monte Carlo algorithm.

The heuristic for remembering which class is which is that the names were chosen to appeal to English speakers: in Las Vegas, the dealer can tell you whether you've won or lost, but in Monte Carlo, *le croupier ne parle que Français*, so you have no idea what he's saying.

Generally, we prefer Las Vegas algorithms, because we like knowing when we have succeeded. But sometimes we have to settle for Monte Carlo algorithms, which can still be useful if we can get the probability of failure small enough. For example, any time we try to estimate an average by **sampling** (say, inputs to a function we are trying to integrate or political views of voters we are trying to win over) we are running a Monte Carlo algorithm: there is always some possibility that our sample is badly non-representative, but we can't tell if we got a bad sample unless we already know the answer we are looking for.

1.5.2 Randomized complexity classes

Las Vegas vs Monte Carlo is the typical distinction made by algorithm designers, but complexity theorists have developed more elaborate classifications. These include algorithms with "one-sided" failure properties. For

[7] To be more precise, Babai defined *Monte Carlo algorithms* based on the properties of **Monte Carlo simulation**, a technique dating back to the Manhattan project. The term *Las Vegas algorithm* was new.

these algorithms, we never get a bogus "yes" answer but may get a bogus "no" answer (or vice versa). This gives us several complexity classes that act like randomized versions of **NP**, co-**NP**, etc.:

- The class **R** or **RP** (randomized **P**) consists of all languages L for which a polynomial-time Turing machine M exists such that if $x \in L$, then $\Pr[M(x,r) = 1] \geq 1/2$ and if $x \notin L$, then $\Pr[M(x,r) = 1] = 0$. In other words, we can find a witness that $x \in L$ with constant probability. This is the randomized analog of **NP** (but it's much more practical, since with **NP** the probability of finding a winning witness may be exponentially small).

- The class co-**R** consists of all languages L for which a poly-time Turing machine M exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \geq 1/2$ and if $x \in L$, then $\Pr[M(x,r) = 1] = 0$. This is the randomized analog of co-**NP**.

- The class **ZPP** (zero-error probabilistic P) is defined as **RP** $\cap$ co-**RP**. If we run both our **RP** and co-**RP** machines for polynomial time, we learn the correct classification of x with probability at least $1/2$. The rest of the time we learn only that we've failed (because both machines return 0, telling us nothing). This is the class of (polynomial-time) Las Vegas algorithms. The reason it is called "zero-error" is that we can equivalently define it as the problems solvable by machines that always output the correct answer eventually, but only run in *expected* polynomial time.

- The class **BPP** (bounded-error probabilistic **P**) consists of all languages L for which a poly-time Turing machine exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \leq 1/3$, and if $x \in L$, then $\Pr[M(x,r) = 1] \geq 2/3$. These are the (polynomial-time) Monte Carlo algorithms: if our machine answers 0 or 1, we can guess whether $x \in L$ or not, but we can't be sure.

- The class **PP** (probabilistic **P**) consists of all languages L for which a poly-time Turing machine exists such that if $x \notin L$, then $\Pr[M(x,r) = 1] \geq 1/2$, and if $x \in L$, then $\Pr[M(x,r) = 1] < 1/2$. Since there is only an exponentially small gap between the two probabilities, such algorithms are not really useful in practice; **PP** is mostly of interest to complexity theorists.

Assuming we have a source of random bits, any algorithm in **RP**, co-**RP**, **ZPP**, or **BPP** is good enough for practical use. We can usually even get away with using a pseudorandom number generator, and there are plausible reasons to suspect that in fact every one of these classes is equal to **P**.

1.6 Classifying randomized algorithms by their methods

We can also classify randomized algorithms by how they use their randomness to solve a problem. Some very broad categories:[8]

- **Avoiding worst-case inputs**, by hiding the details of the algorithm from the adversary. Typically we assume that an adversary supplies our input. If the adversary can see what our algorithm is going to do (for example, he knows which door we will open first), he can use this information against us. By using randomness, we can replace our predictable deterministic algorithm by what is effectively a random choice of many different deterministic algorithms. Since the adversary doesn't know which algorithm we are using, he can't (we hope) pick an input that is bad for all of them.

- **Sampling**. Here we use randomness to find an example or examples of objects that are likely to be typical of the population they are drawn from, either to estimate some average value (pretty much the basis of all of statistics) or because a typical element is useful in our algorithm (for example, when picking the pivot in QuickSort). Randomization means that the adversary can't direct us to non-representative samples.

- **Hashing**. Hashing is the process of assigning a large object x a small name $h(x)$ by feeding it to a **hash function** h. Because the names are small, the Pigeonhole Principle implies that many large objects hash to the same name (a **collision**). If we have few objects that we actually care about, we can avoid collisions by choosing a hash function that happens to map them to different places. Randomization helps here by keeping the adversary from choosing the objects after seeing what our hash function is.

  Hashing techniques are used both in **load balancing** (e.g., insuring that most cells in a **hash table** hold only a few objects) and in

[8] These are largely adapted from the introduction to [MR95].

fingerprinting (e.g., using a **cryptographic hash function** to record a **fingerprint** of a file, so that we can detect when it has been modified).

- **Building random structures**. The **probabilistic method** shows the existence of structures with some desired property (often graphs with interesting properties, but there are other places where it can be used) by showing that a randomly-generated structure in some class has a nonzero probability of having the property we want. If we can beef the probability up to something substantial, we get a randomized algorithm for generating these structures.
- **Symmetry breaking**. In **distributed algorithms** involving multiple processes, progress may be stymied by all the processes trying to do the same thing at the same time (this is an obstacle, for example, in **leader election**, where we want only one process to declare itself the leader). Randomization can break these deadlocks.

Chapter 2

Probability theory

In this chapter, we summarize the parts of **probability theory** that we will use in the course. This is not really a substitute for reading an actual probability theory book like Feller [Fel68] or Grimmett and Stirzaker [GS01], but the hope is that it's enough to get by.

The basic idea of probability theory is that we want to model all possible outcomes of whatever process we are studying simultaneously. This gives the notion of a **probability space**, which is the set of all possible outcomes; for example, if we roll two dice, the probability space would consist of all 36 possible combinations of values. Subsets of this space are called **events**; an example in the two-dice space would be the event that the sum of the two dice is 11, given by the set $A = \{\langle 5, 6\rangle, \langle 6, 5\rangle\}$. The probability of an event A is given by a **probability measure** $\Pr[A]$; for simple probability spaces, this is just the sum of the probabilities of the individual outcomes contains in A, while for more general spaces, we define the measure on events first and the probabilities of individual outcomes are derived from this. Formal definitions of all of these concepts are given later in this chapter.

When analyzing a randomized algorithm, the probability space describes all choices of the random bits used by the algorithm, and we can think of the possible executions of an algorithm as living within this probability space. More formally, the sequence of operations carried out by an algorithm and the output it ultimately produces are examples of **random variables**—functions from a probability space to some other set—which we will discuss in detail in Chapter 3.

2.1 Probability spaces and events

A **discrete probability space** is a countable set Ω of **points** or **outcomes** ω. Each ω in Ω has a **probability** $\Pr[\omega]$, which is a real value with $0 \leq \Pr[\omega] \leq 1$. It is required that $\sum_{\omega \in \Omega} = 1$.

An **event** A is a subset of Ω; its probability is $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$. We require that $\Pr[\Omega] = 1$, and it is immediate from the definition that $\Pr[\emptyset] = 0$.

The **complement** $\bar{A}$ or $\neg A$ of an event A is the event $\Omega - A$. It is always the case that $\Pr[\neg A] = 1 - \Pr[A]$.

This fact is a special case of the general principle that if $A_1, A_2, \ldots$ forms a **partition** of Ω—that is, if $A_i \cap A_j = \emptyset$ when $i \neq j$ and $\bigcup A_i = \Omega$—then $\sum \Pr[A_i] = \Pr[\Omega] = 1$. It happens that $\neg A$ and A form a partition of Ω consisting of exactly two elements.

Even more generally, if $A_1, A_2, \ldots$ are **disjoint** events (that is, if $A_i \cap A_j = \emptyset$ whenever $i \neq j$), then $\Pr[\bigcup A_i] = \sum \Pr[A_i]$. This fact does not hold in general for events that are not disjoint.

For discrete probability spaces, all of these facts can be proven directly from the definition of probabilities for events. For more general probability spaces, it's no longer possible to express the probability of an event as the sum of the probabilities of its elements, and we adopt an axiomatic approach instead.

2.1.1 General probability spaces

More general probability spaces consist of a triple $(\Omega, \mathcal{F}, \Pr)$ where Ω is a set of points, $\mathcal{F}$ is a **σ-algebra** (a family of subsets of Ω that contains Ω and is closed under complement and countable unions) of **measurable sets**, and Pr is a function from $\mathcal{F}$ to $[0, 1]$ that gives $\Pr[\Omega] = 1$ and satisfies **countable additivity**: when $A_1, \ldots$ are disjoint, $\Pr[\bigcup A_i] = \sum \Pr[A_i]$. This definition is needed for uncountable spaces, because (under certain set-theoretic assumptions) we may not be able to assign a meaningful probability to all subsets of Ω.

Formally, this definition is often presented as three **axioms of probability**, due to Kolmogorov [Kol33]:

1. $\Pr[A] \geq 0$ for all $A \in \mathcal{F}$.
2. $\Pr[\Omega] = 1$.

3. For any countable collection of disjoint events $A_1, A_2, \ldots$,

$$\Pr\left[\bigcup_i A_i\right] = \sum_i \Pr\left[A_i\right].$$

It's not hard to see that the discrete probability spaces defined in the preceding section satisfy these axioms.

General probability spaces arise in randomized algorithms when we have an algorithm that might consume an unbounded number of random bits. The problem now is that an outcome consists of countable sequence of bits, and there are uncountably many such outcomes. The solution is to consider as measurable events only those sets with the property that membership in them can be determined after a finite amount of time. Formally, the probability space Ω is the set $\{0,1\}^{\mathbb{N}}$ of all countably infinite sequences of 0 and 1 values indexed by the natural numbers, and the measurable sets $\mathcal{F}$ are all sets that can be generated by countable unions[1] of **cylinder sets**, where a cylinder set consists of all extensions xy of some finite prefix x. The probability measure itself is obtained by assigning the set of all points that start with x the probability $2^{-|x|}$, and computing the probabilities of other sets from the axioms.[2]

An oddity that arises in general probability spaces is it may be that every particular outcome has probability zero but their union has probability 1. For example, the probability of any particular infinite string of bits is 0, but the set containing all such strings is the entire space and has probability 1. This is where the fact that probabilities only add over *countable* unions comes in.

Most randomized algorithms books gloss over general probability spaces, with three good reasons. The first is that if we truncate an algorithm after a finite number of steps, we are usually get back to a discrete probability space, which avoids a lot of worrying about measurability and convergence. The second is that we are often implicitly working in a probability space that is either discrete or well-understood (like the space of bit-vectors described above). The last is that the **Kolmogorov extension theorem** says that

[1] As well as complements and countable intersections. However, it is not hard to show that that sets defined using these operations can be reduced to countable unions of cylinder sets.

[2] This turns out to give the same probabilities as if we consider each outcome as a real number in the interval $[0,1]$ and use Lebesgue measure to compute the probability of events. For some applications, thinking of our random values as real numbers (or even sequences of real numbers) can make things easier: consider for example what happens when we want to choose one of three outcomes with equal probability.

if we specify $\Pr[A_1 \cap A_2 \cap \cdots \cap A_k]$ consistently for all finite sets of events $\{A_1 \ldots A_k\}$, then there exists some probability space that makes these probabilities work, even if we have uncountably many such events. So it's usually enough to specify how the events we care about interact, without worrying about the details of the underlying space.

2.2 Boolean combinations of events

Even though events are defined as sets, we often think of them as representing propositions that we can combine using the usual Boolean operations of NOT ($\neg$), AND ($\wedge$), and OR ($\vee$). In terms of sets, these correspond to taking a complement $\bar{A} = \Omega \setminus A$, an intersection $A \cap B$, or a union $A \cup B$.

We can use the axioms of probability to calculate the probability of $\bar{A}$:

Lemma 2.2.1.

$$\Pr\left[\bar{A}\right] = 1 - \Pr[A].$$

Proof. First note that $A \cap \bar{A} = \emptyset$, so $A \cup \bar{A} = \Omega$ is a disjoint union of countably many[3] events. This gives $\Pr[A] + \Pr\left[\bar{A}\right] = \Pr[\Omega] = 1$. □

For example, if our probability space consists of the six outcomes of a fair die roll, and $A = [\text{outcome is 3}]$ with $\Pr[A] = 5/6$, then $\Pr[\text{outcome is not 3}] = \Pr\left[\bar{A}\right] = 1 - 1/6 = 5/6$. Though this example is trivial, using the formula does save us from having to add up the five cases where we don't get 3.

If we want to know the probability of $A \cap B$, we need to know more about the relationship between A and B. For example, it could be that A and B are both events representing a fair coin coming up heads, with $\Pr[A] = \Pr[B] = 1/2$. The probability of $A \cap B$ could be anywhere between $1/2$ and 0:

- For ordinary fair coins, we'd expect that half the time that A happens, B also happens. This gives $\Pr[A \cap B] = (1/2) \cdot (1/2) = 1/4$. To make this formal, we might define our probability space Ω as having four outcomes HH, HT, TH, and TT, each of which occurs with equal probability.
- But maybe A and B represent the same fair coin: then $A \cap B = A$ and $\Pr[A \cap B] = \Pr[A] = 1/2$.

[3] Countable need not be infinite, so 2 is countable.

- At the other extreme, maybe A and B represent two fair coins welded together so that if one comes up heads the other comes up tails. Now $\Pr[A \cap B] = 0$.
- With a little bit of tinkering, we could also find probabilities for the outcomes in our four-outcome space to make $\Pr[A] = \Pr[\mathsf{HH}] + \Pr[\mathsf{HT}] = 1/2$ and $\Pr[B] = \Pr[\mathsf{HH}] + \Pr[\mathsf{TH}] = 1/2$ while setting $\Pr[A \cap B] = \Pr[\mathsf{HH}]$ to any value between 0 and 1/2.

The difference between the nice case where $\Pr[A \cap B]$ equals 1/4 and the other, more annoying cases where it doesn't is that in the first case we have assumed that A and B are **independent**, which is *defined* to mean that $\Pr[A \cap B] = \Pr[A]\Pr[B]$.

In the real world, we expect events to be independent if they refer to parts of the universe that are not causally related: if we flip two coins that aren't glued together somehow, then we assume that the outcomes of the coins are independent. But we can also get independence from events that are not causally disconnected in this way. An example would be if we rolled a fair four-sided die labeled $\mathsf{HH}, \mathsf{HT}, \mathsf{TH}, \mathsf{TT}$, where we take the first letter as representing A and the second as B.

There's no simple formula for $\Pr[A \cup B]$ when A and B are not disjoint, even for independent events, but we can compute the probability by splitting up into smaller, disjoint events and using countable additivity:

$$\begin{aligned}
\Pr[A \cup B] &= \Pr\left[(A \cap B) \cup (A \cap \bar{B}) \cup (\bar{A} \cap B)\right] \\
&= \Pr[A \cap B] + \Pr\left[A \cap \bar{B}\right] + \Pr\left[\bar{A} \cap B\right] \\
&= \left(\Pr[A \cap B] + \Pr\left[A \cap \bar{B}\right]\right) + \left(\Pr\left[\bar{A} \cap B\right] + \Pr[A \cap B]\right) - \Pr[A \cap B] \\
&= \Pr[A] + \Pr[B] - \Pr[A \cap B].
\end{aligned}$$

The idea is that we can compute $\Pr[A \cup B]$ by adding up the individual probabilities and then subtracting off the part where the counted the event twice.

This is a special case of the general **inclusion-exclusion formula**, which says:

Lemma 2.2.2. *For any finite sequence of events $A_1 \dots A_n$,*

$$\begin{aligned}\Pr\left[\bigcup_{i=1}^{n} A_i\right] &= \sum_i \Pr[A_i] - \sum_{i<j} \Pr[A_i \cap A_j] + \sum_{i<j<k} \Pr[A_i \cap A_j \cap A_k] - \dots \\ &= \sum_{S\subseteq\{1\dots n\}, S\neq\emptyset} (-1)^{|S|+1} \Pr\left[\bigcap_{i\in S} A_i\right]. \end{aligned} \tag{2.2.1}$$

Proof. Partition Ω into 2^n disjoint events B_T, where $B_T = \left(\bigcap_{i\in T} A_i\right) \cap \left(\bigcap_{i\notin T} \bar{A}_i\right)$ is the event that all A_i occur for i in T and no A_i occurs for i not in T. Then A_i is the union of all B_T with $T \ni i$ and $\bigcup A_i$ is the union of all B_T with $T \neq \emptyset$.

That the right-hand side gives the probability of this event is a sneaky consequence of the binomial theorem, and in particular the fact that $\sum_{i=1}^{n}(-1)^i = \sum_{i=0}^{n}(-1)^i - 1 = (1-1)^n - 1$ is -1 if $n > 0$ and 0 if $n = 0$. Using this fact after rewriting the right-hand side using the B_T events gives

$$\begin{aligned}\sum_{S\subseteq\{1\dots n\}, S\neq\emptyset} (-1)^{|S|+1} \Pr\left[\bigcap_{i\in S} A_i\right] &= \sum_{S\subseteq\{1\dots n\}, S\neq\emptyset} (-1)^{|S|+1} \sum_{T\supseteq S} \Pr[B_T] \\ &= \sum_{T\subseteq\{1\dots n\}} \left(\Pr[B_T] \sum_{S\subseteq T, S\neq\emptyset} (-1)^{|S|+1}\right) \\ &= \sum_{T\subseteq\{1\dots n\}} \left(-\Pr[B_T] \sum_{i=1}^{n} (-1)^i \binom{|T|}{i}\right) \\ &= \sum_{T\subseteq\{1\dots n\}} \left(-\Pr[B_T]\left((1-1)^{|T|} - 1\right)\right) \\ &= \sum_{T\subseteq\{1\dots n\}} \Pr[B_T]\left(\left(1 - 0^{|T|}\right)\right) \\ &= \sum_{T\subseteq\{1\dots n\}, T\neq\emptyset} \Pr[B_T] \\ &= \Pr\left[\bigcup_{i=1}^{n} A_i\right]. \end{aligned}$$

□

2.3 Conditional probability

The **probability of A conditioned on B** or **probability of A given B**, written $\Pr[A \mid B]$, is defined by

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}, \tag{2.3.1}$$

provided $\Pr[B \neq 0]$. If $\Pr[B] = 0$, we can't condition on B.

Such conditional probabilities represent the effect of restricting our probability space to just B, which can think of as computing the probability of each event if we know that B occurs. The intersection in the numerator limits A to circumstances where B occurs, while the denominator normalizes the probabilities so that, for example, $\Pr[\Omega \mid B] = \Pr[B \mid B] = 1$.

2.3.1 Conditional probability and independence

Rearranging (2.3.1) gives $\Pr[A \cap B] = \Pr[B]\Pr[A \mid B] = \Pr[A]\Pr[B \mid A]$. In many cases, knowing that B occurs tells us nothing about whether A occurs; if so, we have $\Pr[A \mid B] = \Pr[B]$, which implies that $\Pr[A \cap B] = \Pr[A \mid B]\Pr[B] = \Pr[A]\Pr[B]$—events A and B are **independent**. So $\Pr[A \mid B] = \Pr[A]$ gives an alternative criterion for independence when $\Pr[B]$ is nonzero.[4]

A *set* of events $A_1, A_2, \ldots$ is independent if A_i is independent of B when B is any Boolean formula of the A_j for $j \neq i$. The idea is that you can't predict A_i by knowing anything about the rest of the events.

A set of events $A_1, A_2, \ldots$ is **pairwise independent** if each A_i and A_j are independent when $i \neq j$. It is possible for a set of events to be pairwise independent but not independent; a simple example is when A_1 and A_2 are the events that two independent coins come up heads and A_3 is the event that both coins come up with the same value. The general version of pairwise independence is **k-wise independence**, which means that any subset of k (or fewer) events are independent.

2.3.2 Conditional probability and the law of total probability

The reason we like conditional probability in algorithm analysis is that it gives us a natural way to model the kind of case analysis that we are used to applying to deterministic algorithms. Suppose we are trying to prove that a randomized algorithm works (event A) with a certain probability. Most

[4] If $\Pr[B]$ is zero, then A and B are always independent.

likely, the first random thing the algorithm does is flip a coin, giving two possible outcomes B and $\bar{B}$. Countable additivity tells us that $\Pr[A] = \Pr[A \cap B] + \Pr\left[A \cap \bar{B}\right]$, which we can rewrite using conditional probability as

$$\Pr[A] = \Pr[A \mid B] \Pr[B] + \Pr\left[A \;\middle|\; \bar{B}\right] \Pr\left[\bar{B}\right], \tag{2.3.2}$$

a special case of the **law of total probability**.

What's nice about this expression is that we can often compute $\Pr[A \mid B]$ and $\Pr\left[A \;\middle|\; \bar{B}\right]$ by looking at what the algorithm does starting from the point where it has just gotten heads (B) or tails ($\bar{B}$), and use the formula to combine these values to get the overall probability of success.

For example, if

$$\begin{aligned} \Pr[\text{class occurs} \mid \text{snow}] &= 3/5, \\ \Pr[\text{class occurs} \mid \text{no snow}] &= 99/100, \text{and} \\ \Pr[\text{snow}] &= 1/10, \end{aligned}$$

then

$$\Pr[\text{class occurs}] = (3/5) \cdot (1/10) + (99/100) \cdot (1 - 1/10) \quad = 0.951.$$

More generally, we can do the same computation for any partition of Ω into countably many disjoint events B_i:

$$\begin{aligned} \Pr[A] &= \Pr\left[\bigcup_i (A \cap B_i)\right] \\ &= \sum_i \Pr[A \cap B_i] \\ &= \sum_{i, \Pr[B_i] \neq 0} \Pr[A \mid B_i] \Pr[B_i], \end{aligned} \tag{2.3.3}$$

which is the **law of total probability**. Note that the last step works for each term only if $\Pr[A \mid B_i]$ is well-defined, meaning that $\Pr[B_i] \neq 0$. But any case where $\Pr[B_i] = 0$ also has $\Pr[A \cap B_i] = 0$, so we get the correct answer if we simply omit these terms from both sums.

A special case arises when $\Pr\left[A \;\middle|\; \bar{B}\right] = 0$, which occurs, for example, if $A \subseteq B$. Then we just have $\Pr[A] = \Pr[A \mid B] \Pr[B]$. If we consider an

event $A = A_1 \cap A_2 \cap \cdots \cap A_k$, then we can iterate this expansion to get

$$\begin{aligned}\Pr[A_1 \cap A_2 \cap \cdots \cap A_k] &= \Pr[A_1 \cap \cdots \cap A_{k-1}] \Pr[A_k \mid A_1, \ldots, A_{k-1}] \\ &= \Pr[A_1 \cap \cdots \cap A_{k-2}] \Pr[A_{k-1} \mid A_1, \ldots, A_{k-2}] \Pr[A_k \mid A_1, \ldots, A_{k-1}] \\ &= \ldots \\ &= \prod_{i=1}^{k} \Pr[A_i \mid A_1, \ldots, A_i]. \end{aligned} \tag{2.3.4}$$

Here $\Pr[A \mid B, C, \ldots]$ is short-hand for $\Pr[B \cap C \cap \ldots]$, the probability that A occurs given that all of B, C, etc., occur.

2.3.3 Examples

Here we have some examples of applying conditional probability to algorithm analysis. Mostly we will be using some form of the law of total probability.

2.3.3.1 Racing coin-flips

Suppose that I flip coins and allocate a space for each heads that I get before the coin comes up tails. Suppose that you then supply me with objects (each of which takes up one unit of space), one for each heads that *you* get before you get tails. What are my chances of allocating enough space?

Let's start by solving this directly using the law of total probability. Let A_i be the event that I allocate i spaces. The event A_i is the intersection of i independent events that I get heads in the first i positions and the event that I get tails in position $i+1$; this multiplies out to $(1/2)^{i+1}$. Let B_i be the similar event that you supply i objects. Let W be the event that I win. To make the A_i partition the space, we must also add an extra event A_∞ equal to the singleton set $\{\mathsf{HHHHHHH}\ldots\}$ consisting of the all-H sequence; this has probability 0 (so it won't have much of an effect), but we need to include it since $\mathsf{HHHHHHH}\ldots$ is not contained in any of the other A_i.

We can compute

$$\begin{aligned}
\Pr[W \mid A_i] &= \Pr[B_0 \cap B_1 \cap \cdots \cap B_i \mid A_i] \\
&= \Pr[B_0 \cap B_1 \cap \cdots \cap B_i] \\
&= \Pr[B_0] + \Pr[B_1] + \cdots + \Pr[B_i] \\
&= \sum_{j=0}^{i} (1/2)^i \\
&= (1/2) \cdot \frac{1 - (1/2)^{i+1}}{1 - 1/2} \\
&= 1 - (1/2)^{i+1}. \qquad (2.3.5)
\end{aligned}$$

The clean form of this expression suggests strongly that there is a better way to get it, and that this way involves taking the negation of the intersection of $i+1$ independent events that occur with probability $1/2$ each. With a little reflection, we can see that the probability that your objects *don't* fit in my buffer is exactly $(1/2)^{i+1}$

From the law of total probability (2.3.3),

$$\begin{aligned}
\Pr[W] &= \sum_{i=0}^{\infty} (1 - (1/2)^{i+1})(1/2)^{i+1} \\
&= 1 - \sum_{i=0}^{\infty} (1/4)^{i+1} \\
&= 1 - \frac{1}{4} \cdot \frac{1}{1 - 1/4} \\
&= 2/3.
\end{aligned}$$

This gives us our answer. However, we again see an answer that is suspiciously simple, which suggests looking for another way to find it. We can do this using conditional probability by defining new events C_i, where C_i contains all sequences of coin-flips for both players where get i heads in a row but at least one gets tails on the $(i+1)$-th coin. These events plus the probability-zero event $C_\infty = \{\mathsf{HHHHHHH}\ldots, \mathsf{TTTTTTT}\ldots\}$ partition the space, so $\Pr[W] = \sum_{i=0}^{\infty} \Pr[W \mid C_i] \Pr[C_i]$.

Now we ask, what is $\Pr[W \mid C_i]$? Here we only need to consider three cases, depending on the outcomes of our $(i+1)$-th coin-flips. The cases $\langle \mathsf{H}, \mathsf{T} \rangle$ and $\langle \mathsf{T}, \mathsf{T} \rangle$ cause me to win, while the case $\langle \mathsf{T}, \mathsf{H} \rangle$ causes me to lose, and each case occurs with equal probability conditioned on C_i (which excludes $\langle \mathsf{H}, \mathsf{H} \rangle$). So I win $2/3$ of the time conditioned on C_i, and summing

$\Pr[W] = \sum_{i=0}^{\infty}(2/3)\Pr[C_i] = 2/3$ since $\Pr[C_i]$ sums to 1, because the union of these events includes the entire space except for the probability-zero event C_∞.

Still another approach is to compute the probability that our runs have exactly the same length ($\sum_{i=1}^{\infty} 2^{-i} \cdot 2^{-i} = 1/3$), and argue by symmetry that the remaining 2/3 probability is equally split between my run being longer (1/3) and your run being longer (1/3). Since W occurs if my run is just as long or longer, $\Pr[W] = 1/3 + 1/3 = 2/3$. A nice property of this approach is that the only summation involved is over disjoint events, so we get to avoid using conditional probability entirely.

2.3.3.2 Karger's min-cut algorithm

Here we'll give a simple algorithm for finding a global **min-cut** in a **multigraph**,[5] due to David Karger [Kar93].

The idea is that we are given a multigraph G, and we want to partition the vertices into nonempty sets S and T such that the number of edges with one endpoint in S and one endpoint in T is as small as possible. There are many efficient ways to do this, most of which are quite sophisticated. There is also the algorithm we will now present, which solves the problem with reasonable efficiency using almost no sophistication at all (at least in the algorithm itself).

The main idea is that given an edge uv, we can construct a new multigraph G_1 by **contracting** the edge: in G_1, u and v are replaced by a single vertex, and any edge that used to have either vertex as an endpoint now goes to the combined vertex (edges with both endpoints in $\{u, v\}$ are deleted). Karger's algorithm is to contract edges chosen uniformly at random until only two vertices remain. All the vertices that got packed into one of these become S, the others become T. It turns out that this finds a minimum cut with probability at least $1/\binom{n}{2}$.

An example of the algorithm in action is given in Figure 2.1.

Theorem 2.3.1. *Given any min cut (S,T) of a graph G on n vertices, Karger's algorithm outputs (S,T) with probability at least $1/\binom{n}{2}$.*

Proof. Let (S,T) be a min cut of size k. Then the degree of each vertex v is at least k (otherwise $(v, G-v)$ would be a smaller cut), and G contains at least $kn/2$ edges. The probability that we contract an S–T edge is thus at most $k/(kn/2) = 2/n$, and the probability that we don't contract one is

[5] Unlike ordinary graphs, multigraphs can have more than one edge between two vertices.

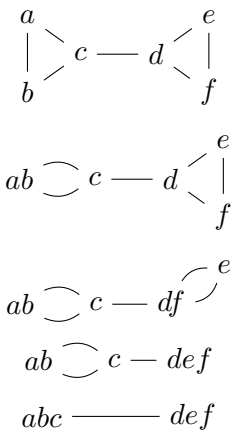


Figure 2.1: Karger's min-cut algorithm. Initial graph (at top) has min cut $\langle \{a, b, c\}, \{d, e, f\} \rangle$. We find this cut by getting lucky and contracting edges ab, df, de, and ac in that order. The final graph (at bottom) gives the cut.

$1 - 2/n = (n-2)/n$. Assuming we missed collapsing (S, T) the first time, we now have a new graph G_1 with $n-1$ vertices in which the min cut is still of size k. So now the chance that we miss (S, T) is $(n-3)/(n-1)$. We stop when we have two vertices left, so the last step succeeds with probability $1/3$.

We can compute the probability that the S–T cut is never contracted by applying (2.3.4), which just tells us to multiply all the conditional probabilities together:

$$\prod_{i=3}^{n} \frac{i-2}{i} = \frac{2}{n(n-1)}.$$

□

This tells us what happens when we are considering a particular min cut. If the graph has more than one min cut, this only makes our life easier. Note that since each min cut turns up with probability at least $1/\binom{n}{2}$, there can't be more than $\binom{n}{2}$ of them.[6] But even if there is only one, we have a good

[6]The suspiciously combinatorial appearance of the $1/\binom{n}{2}$ suggests that there should be some way of associating minimum cuts with particular pairs of vertices, but I'm not aware of any natural way to do this. Sometimes the appearance of a simple expression in a surprising context may just stem from the fact that there aren't very many distinct simple expressions.

chance of finding it if we simply re-run the algorithm substantially more than n^2 times.

Chapter 3

Random variables

Probabilities are fine when all we want to do is ask whether an algorithm worked or not. But for many randomized algorithms (particularly Las Vegas algorithms), we can structure the algorithm so that it works eventually with probability 1. This makes the important question that of how long is eventually. To measure a quantity like running time that depends on the random choices of the algorithm, we need a **random variable**, which is just a function whose domain is some probability space Ω.[1]

Even though random variables are just functions, rather than writing a random variable as $f(\omega)$ everywhere, the convention is to write a random variable as a capital letter (X, Y, S, etc.) and make the argument implicit: X is really $X(\omega)$. Variables that aren't random (or aren't variable) are written in lowercase.

Most of the random variables we will consider will be **discrete random variables**. A discrete random variable takes on only countably many values, each with some nonzero probability.

For example, consider the probability space corresponding to rolling two independent fair six-sided dice. There are 36 possible outcomes in this space, corresponding to the 6×6 pairs of values $\langle x, y \rangle$ we might see on the two dice. We could represent the value of each die as a random variable X or Y given by $X(\langle x, y \rangle) = x$ or $Y(\langle x, y \rangle) = y$, but for many applications, we don't care so much about the specific values on each die. Instead, we want to know the sum $S = X + Y$ of the dice. This value S is also random variable; as a function on Ω, it's defined by $S(\langle x, y \rangle) = x + y$.

[1] Technically, this only works for discrete spaces. In general, a random variable is a **measurable function** from a probability space $(\Omega, \mathcal{F})$ to some other set S equipped with its own σ-algebra $\mathcal{F}'$. What makes a function measurable in this sense is that that for any set A in $\mathcal{F}'$, the inverse image $f^{-1}(A)$ must be in $\mathcal{F}$. See §3.3 for more details.

Random variables need not be real-valued. There's no reason why we can't think of the pair $\langle x, y \rangle$ itself a random variable, whose range is the set $[1 \dots 6] \times [1 \dots 6]$. Similarly, if we imagine choosing a point uniformly at random in the unit square $[0, 1]^2$, its coordinates are a random variable. For a more exotic example, the **random graph** $G_{n,p}$ obtained by starting with n vertices and including each possible edge with independent probability p is a random variable whose range is the set of all graphs on n vertices.

3.1 Operations on random variables

Random variables may be combined using standard arithmetic operators, have functions applied to them, etc., to get new random variables. For example, the random variable X/Y is a function from Ω that takes on the value $X(\omega)/Y(\omega)$ on each point ω.

3.2 Random variables and events

Any random variable X allows us to define events based on its possible values. Typically these are expressed by writing a predicate involving the random variable in square brackets. An example would be the probability that the sum of two dice is exactly 11: $[S = 11]$; or that the sum of the dice is less than 5: $[S < 5]$. These are both sets of outcomes; we could expand $[S = 11] = \{\langle 5, 6 \rangle, \langle 6, 5 \rangle\}$ or $[S < 5] = \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 1 \rangle, \langle 2, 2 \rangle, \langle 3, 1 \rangle\}$. This allows us to calculate the probability that a random variable has particular properties: $\Pr[S = 11] = \frac{2}{36} = \frac{1}{18}$ and $\Pr[S < 5] = \frac{6}{36} = \frac{1}{6}$.

Conversely, given any event A, we can define an **indicator random variable** 1_A that is 1 when A occurs and 0 when it doesn't.[2] Formally, $1_A(\omega) = 1$ for ω in A and $1_A(\omega) = 0$ for ω not in A.

Indicator variables are mostly useful when combined with other random variables. For example, if you roll two dice and normally collect the sum of the values but get nothing if it is 7, we could write your payoff as $S \cdot 1_{[S \neq 7]}$.

The **probability mass function** of a random variable gives $\Pr[X = x]$ for each possible value x. For example, our random variable S has the

[2] Some people like writing χ_A for these.

You may also see $[P]$ where P is some predicate, a convention known as **Iverson notation** or the **Iverson bracket** that was invented by Iverson for the programming language APL, appears in later languages like *C* where the convention is that true predicates evaluate to 1 and false ones to 0, and ultimately popularized for use in mathematics—with the specific choice of square brackets to set off the predicate—by Graham *et al.* [GKP88].

Out of these alternatives, I personally find 1_A to be the least confusing.

S	Probability
2	1/36
3	2/36
4	3/36
5	4/36
6	5/36
7	6/36
8	5/36
9	4/36
10	3/36
11	2/36
12	1/36

Table 3.1: Probability mass function for the sum of two independent fair six-sided dice

probability mass function show in Table 3.1. For a discrete random variable X, the probability mass function gives enough information to calculate the probability of any event involving X, since we can just sum up cases using countable additivity. This gives us another way to compute $\Pr[S < 5] = \Pr[S = 2] + \Pr[S = 3] + \Pr[S = 4] = \frac{1+2+3}{36} = \frac{1}{6}$.

For two random variables, the **joint probability mass function** gives $\Pr[X = x \wedge Y = y]$ for each pair of values x and y. This generalizes in the obvious way for more than two variables.

We will often refer to the probability mass function as giving the **distribution** or **joint distribution** of a random variable or collection of random variables, even though distribution (for real-valued variables) technically refers to the **cumulative distribution function** $F(x) = \Pr[X \leq x]$, which is generally not directly computable from the probability mass function for **continuous random variables** that take on uncountably many values. To the extent that we can, we will try to avoid continuous random variables, and the rather messy integration theory needed to handle them.

Two or more random variables are **independent** if all sets of events involving different random variables are independent. In terms of proba-

bility mass functions, X and Y are independent if $\Pr[X = x \wedge Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$ for any constants x and y. In terms of cumulative distribution functions, X and Y are independent if $\Pr[X \leq x \wedge Y \leq y] = \Pr[X = x] \cdot \Pr[Y = y]$ for any constants x and y. As with events, we generally assume that random variables associated with causally disconnected processes are independent, but this is not the only way we might have independence.

It's not hard to see that the individual die values X and Y in our two-dice example are independent, because every possible combination of values x and y has the same probability $1/36 = \Pr[X = x] \Pr[Y = y]$. If we chose a different probability distribution on the space, we might not have independence.

3.3 Measurability

For discrete probability spaces, any function on outcomes can be a random variable. The reason is that any event in a discrete probability space has a well-defined probability. For more general spaces, in order to be useful, events involving a random variable should have well-defined probabilities. For **discrete random variables** that take on only countably many values (e.g., integers or rationals), it's enough for the event $[X = x]$ (that is, the set $\{\omega \mid X(\omega) = x\}$) to be in $\mathcal{F}$ for all x. For real-valued random variables, we ask that the event $[X \leq x]$ be in $\mathcal{F}$. In these cases, we say that X is **measurable** with respect to $\mathcal{F}$, or just **measurable** $\mathcal{F}$. More exotic random variables use a definition of measurability that generalizes the real-valued version, which we probably won't need.[3] Since we usually just assume that all of our random variables are measurable unless we are doing something funny with $\mathcal{F}$ to represent ignorance, this issue won't come up much.

3.4 Expectation

The **expectation** or **expected value** of a random variable X is given by $\mathrm{E}[X] = \sum_x x \Pr[X = x]$. This is essentially an average value of X weighted by probability, and it only makes sense if X takes on values that can be summed in this way (e.g., real or complex values, or vectors in a real- or

[3] The general version is that if X takes on values on another measure space $(\Omega', \mathcal{F}')$, then the inverse image $X^{-1}(A) = \{\omega \in \Omega \mid X(\omega) \in A\}$ of any set A in $\mathcal{F}'$ is in $\mathcal{F}$. This means in particular that $\Pr_\Omega$ maps through X to give a probability measure on Ω' by $\Pr_{\Omega'}[A] = \Pr_\Omega[X^{-1}(A)]$, and the condition on $X^{-1}(A)$ being in $\mathcal{F}$ makes this work.

complex-valued vector space). Even if the expectation makes sense, it may be that a particular random variable X doesn't have an expectation, because the sum fails to converge.[4]

For an example that does work, if X and Y are independent fair six-sided dice, then $\mathrm{E}[X] = \mathrm{E}[Y] = \sum_{k=1}^{6} k\left(\frac{1}{6}\right) = \frac{21}{6} = \frac{7}{2}$, while $\mathrm{E}[X+Y]$ is the rather horrific

$$\sum_{k=2}^{12} k \cdot \Pr[X+Y=i] = \frac{2\cdot 1 + 3\cdot 2 + 4\cdot 3 + 5\cdot 4 + \cdots + 9\cdot 4 + 10\cdot 3 + 11\cdot 2 + 12\cdot 1}{36}$$
$$= \frac{252}{36} = 7.$$

The fact that $7 = \frac{7}{2} + \frac{7}{2}$ here is not a coincidence, but a consequence of **linearity of expectation**, which is the subject of the next section.

3.4.1 Linearity of expectation

The main reason we like expressing the run times of algorithms in terms of expectation is **linearity of expectation**: $\mathrm{E}[aX + bY] = \mathrm{E}[aX] + \mathrm{E}[bY]$ for all random variables X and Y for which $\mathrm{E}[X]$ and $\mathrm{E}[Y]$ are defined, and all constants a and b. This means that we can compute the running time for different parts of our algorithm separately and then add them together, *even if the costs of different parts of the algorithm are not independent.*

The general version is $\mathrm{E}[\sum a_i X_i] = \sum a_i \mathrm{E}[X_i]$ for any *finite* collection of random variables X_i and constants a_i, which follows by applying induction to the two-variable case. A special case is $\mathrm{E}[cX] = c\,\mathrm{E}[X]$ when c is constant.

For discrete random variables, linearity of expectation follows immediately from the definition of expectation and the fact that the event $[X = x]$ is the

[4]Example: Let X be the number of times you flip a fair coin until it comes up heads. We'll see later than $\mathrm{E}[X] = 1/(1/2) = 2$. But $\mathrm{E}\left[2^X\right] = \sum_{n=1}^{\infty} 2^n 2^{-n} = \sum_{n=1}^{\infty} 1$, which diverges. With some tinkering it is possible to come up with even uglier cases, like an array that contains 1 element on average but requires infinite expected time to sort using a $\Theta(n \log n)$ algorithm.

disjoint union of the events $[X = x, Y = y]$ for all y:

$$\begin{aligned}
\mathrm{E}[aX + bY] &= \sum_{x,y} (ax + by) \Pr[X = x \wedge Y = y] \\
&= a \sum_{x,y} x \Pr[X = x, Y = y] + b \sum_{x,y} y \Pr[X = x, Y = y] \\
&= a \sum_{x} x \sum_{y} Pr[X = x, Y = y] + b \sum_{y} y \sum_{x} \Pr[X = x, Y = y] \\
&= a \sum_{x} x \Pr[X = x] + b \sum_{y} y \Pr[Y = y] \\
&= a \mathrm{E}[X] + b \mathrm{E}[Y].
\end{aligned}$$

A technical note: we are assuming that $\mathrm{E}[X]$, $\mathrm{E}[Y]$, and $\mathrm{E}[X + Y]$ all exist.

This proof does *not* require that X and Y be independent. The sum of two fair six-sided dice always has expectation $\frac{7}{2} + \frac{7}{2} = 7$, whether they are independent dice, the same die counted twice, or one die X and its complement $7 - X$.

Linearity of expectation makes it easy to compute the expectations of random variables that can be expressed as sums of other random variables. One example that will come up a lot is a **binomial random variable**, which is a sum $S = \sum_{i=1}^{n} X_i$ of n independent **Bernoulli random variables**, each of which is 1 with probability p and 0 with probability $q = 1 - p$. These are called binomial random variables because the probability that S is equal to k is given by

$$\Pr[S = k] = \binom{n}{k} p^k q^{n-k}, \tag{3.4.1}$$

which is the k-th term in the binomial expansion of $(p + q)^n$. In this case each X_i has $\mathrm{E}[X_i] = p$, so $\mathrm{E}[S]$ is just np. It is possible to calculate this fact directly from (3.4.1), but it's much more work.[5]

3.4.1.1 Linearity of expectation for infinite sequences

For infinite sequences of random variables, linearity of expectation may break down. This is true even if the sequence is countable. An example

[5] One way is to use the **probability generating function** $F(z) = \sum_{k=0}^{\infty} \Pr[S = k] z^k = \sum_{k=0}^{\infty} \binom{n}{k} p^k q^{n-k} z^k = (pz + q)^n$. Then take the derivative $F'(z) = \sum_{k=0}^{\infty} \Pr[S = k] k z^{k-1}$ and observe $F'(1) = \sum_{k=0}^{\infty} \Pr[S = k] k = \mathrm{E}[S]$. Or we can write $F'(z)$ as $n(pz + q)^{n-1} p$, which gives $F'(1) = n(p + q)^{n-1} p = np$.

is the **St. Petersburg paradox**, in which a gambler bets \$1 on a double-or-nothing game, then bets \$2 if they lose, then \$4, and so on, until they eventually wins and stops, up \$1. If we represent the gambler's gain or loss at stage i as a random variable X_i, it's easy to show that $\mathrm{E}[X_i] = 0$, because the gambler either wins $\pm 2^i$ with equal probability, or doesn't play at all. So $\sum_{i=0}^{\infty} \mathrm{E}[X_i] = 0$. But $\mathrm{E}[\sum_{i=0}^{\infty} X_i] = 1$, because the probability that the gambler doesn't eventually win is zero.[6]

Fortunately, these pathological cases don't come up often in algorithm analysis, and with some additional side constraints we can apply linearity of expectation even to infinite sums of random variables. The simplest is when $X_i \geq 0$ for all i; then $\mathrm{E}[\sum_{i=0}^{\infty} X_i]$ exists and is equal to $\sum_{i=0}^{\infty} \mathrm{E}[X_i]$ whenever the sum of the expectations converges (this is a consequence of the monotone convergence theorem). Another condition that works is if $|\sum_{i=0}^{n} X_i| \leq Y$ for all n, where Y is a random variable with finite expectation; the simplest version of this is when Y is constant. See [GS92, §5.6.12] or [Fel71, §IV.2] for more details.

3.4.2 Expectation and inequalities

If $X \leq Y$ (that is, if the event $[X \leq Y]$ holds with probability 1), then $\mathrm{E}[X] \leq \mathrm{E}[Y]$. For finite discrete spaces the proof is trivial:

$$\begin{aligned}\mathrm{E}[X] &= \sum_{\omega \in \Omega} \Pr[\omega] X(\omega) \\ &\leq \sum_{\omega \in \Omega} \Pr[\omega] Y(\omega) \\ &= \mathrm{E}[Y].\end{aligned}$$

The claim continues to hold even in the general case, but the proof is more work.

One special case of this that comes up often is that $X \geq 0$ implies $\mathrm{E}[X] \geq 0$.

[6] The trick here is that we are trading a probability-1 gain of 1 against a probability-0 loss of ∞. So we could declare that $\mathrm{E}\left[\sum_{i=0}^{\infty} X_i\right]$ involves $0 \cdot (-\infty)$ and is undefined. But this would lose the useful property that expectation isn't affected by probability-0 outcomes. As often happens in mathematics, we are forced to choose between candidate definitions based on which bad consequences we most want to avoid, with no way to avoid all of them. So the standard definition of expectation allows the St. Petersburg paradox because the alternatives are worse.

3.4.3 Expectation of a product

When two random variables X and Y are independent, it also holds that $\mathrm{E}[XY] = \mathrm{E}[X]\,\mathrm{E}[Y]$. The proof (at least for discrete random variables) is straightforward:

$$\begin{aligned}\mathrm{E}[XY] &= \sum_x \sum_y xy \Pr[X = x \wedge Y = y] \\ &= \sum_x \sum_y xy \Pr[X = x] \Pr[Y = y] \\ &= \left(\sum_x x \Pr[X = x]\right)\left(\sum_y \Pr[Y = y]\right) \\ &= \mathrm{E}[X]\,\mathrm{E}[Y]\,.\end{aligned}$$

For example, the expectation of the product of two independent fair six-sided dice is $\left(\frac{7}{2}\right)^2 = \frac{49}{4}$.

This is not true for arbitrary random variables. If we compute the expectation of the product of a single fair six-sided die with itself, we get $\frac{1\cdot1+2\cdot2+3\cdot3+4\cdot4+5\cdot5+6\cdot6}{6} = \frac{91}{6}$, which is much larger.

One measure of the dependence between two random variables by is the difference $\mathrm{E}[XY] - \mathrm{E}[X] \cdot \mathrm{E}[Y]$. This is called the **covariance** of X and Y, written $\mathrm{Cov}[X, Y]$, and it is 0 when X and Y are independent and nonzero otherwise. Covariance will come back later when we look at concentration bounds in Chapter 5.

3.4.3.1 Wald's equation (simple version)

Computing the expectation of a product does not often come up directly in the analysis of a randomized algorithm. Where we might expect to do it is when we have a loop: one random variable N tells us the number of times we execute the loop, while another random variable X tells us the cost of each iteration. The problem is that if each iteration is randomized, then we really have a sequence of random variables $X_1, X_2, \ldots$, and what we want to calculate is

$$\mathrm{E}\left[\sum_{i=1}^{N} X_i\right]. \tag{3.4.2}$$

Here we can't use the sum formula directly, because N is a random variable, and we can't use the product formula, because the X_i are all different random variables.

If N and the X_i are all independent (which may or may not be the case for the loop example), and N is bounded by some fixed maximum n, then we can apply the product rule to get the value of (3.4.2) by throwing in a few indicator variables. The idea is that the contribution of X_i to the sum is given by $X_i 1_{[N \geq i]}$, and because we assume that N is independent of the X_i, if we need to compute $\operatorname{E}\left[X_i 1_{[N \geq i]}\right]$, we can do so by computing $\operatorname{E}[X_i] \operatorname{E}\left[1_{[N \geq i]}\right]$.

So we get

$$\begin{aligned}
\operatorname{E}\left[\sum_{i=1}^{N} X_i\right] &= \operatorname{E}\left[\sum_{i=1}^{n} X_i 1_{[N \geq i]}\right] \\
&= \sum_{i=1}^{n} \operatorname{E}\left[X_i 1_{[N \geq i]}\right] \\
&= \sum_{i=1}^{n} \operatorname{E}[X_i] \operatorname{E}\left[1_{[N \geq i]}\right].
\end{aligned}$$

For general X_i we have to stop here. But if we also know that the X_i all have the same expectation μ, then $\operatorname{E}[X_i]$ doesn't depend on i and we can bring it out of the sum. This gives

$$\begin{aligned}
\sum_{i=1}^{n} \operatorname{E}[X_i] \operatorname{E}\left[1_{[N \geq i]}\right] &= \mu \sum_{i=1}^{n} \operatorname{E}\left[1_{[N \geq i]}\right] \\
&= \mu \operatorname{E}[N]. \qquad (3.4.3)
\end{aligned}$$

This equation is a special case of **Wald's equation**, which we will see again in §9.4.2. The main difference between this version and the general version is that here we had to assume that N was independent of the X_i, which may not be true if our loop is a `while` loop, and termination after a particular iteration is correlated with the time taken by that iteration.

But for simple cases, (3.4.3) can still be useful. For example, if we throw one six-sided die to get N, and then throw N six-sided dice and add them up, we get the same expected total $\frac{7}{2} \cdot \frac{7}{2} = \frac{49}{4}$ as if we just multiply two six-sided dice. This is true even though the actual distribution of the values is very different in the two cases.

3.5 Conditional expectation

We can also define a notion of **conditional expectation**, analogous to conditional probability. There are three versions of this, depending on how

fancy we want to get about specifying what information we are conditioning on.

3.5.1 Expectation conditioned on an event

The expectation of X conditioned on an *event* A is written $\mathrm{E}[X \mid A]$ and defined by

$$\mathrm{E}[X \mid A] = \sum_x x \Pr[X = x \mid A] = \sum_x x \frac{\Pr[X = x \wedge A]}{\Pr[A]}. \tag{3.5.1}$$

This is essentially the weighted average value of X if we know that A occurs.

Most of the properties that we see with ordinary expectations continue to hold for conditional expectation. For example, linearity of expectation

$$\mathrm{E}[aX + bY \mid A] = a\,\mathrm{E}[X \mid A] + b\,\mathrm{E}[Y \mid A] \tag{3.5.2}$$

holds whenever a and b are constant on A.

Similarly if $\Pr[X \geq Y]\, A = 1$, $\mathrm{E}[X \mid A] \geq \mathrm{E}[Y \mid A]$.

Conditional expectation is handy because we can use it to compute expectations by case analysis the same way we use conditional probabilities using the law of total probability (see §2.3.2). If $A_1, A_2, \ldots$ are a countable partition of Ω, then

$$\begin{aligned}
\sum_i \Pr[A_i]\,\mathrm{E}[X \mid A_i] &= \sum_i \Pr[A_i] \left(\sum_x x \Pr[X = x \mid A_i] \right) \\
&= \sum_i \Pr[A_i] \left(\sum_x x \frac{\Pr[X = x \wedge A_i]}{\Pr[A]_i} \right) \\
&= \sum_i \sum_x x \Pr[X = x \wedge A_i] \\
&= \sum_x x \left(\sum_i \Pr[X = x \wedge A_i] \right) \\
&= \sum_x x \Pr[X = x] \\
&= \mathrm{E}[X]. \qquad (3.5.3)
\end{aligned}$$

This is actually a special case of the law of iterated expectation, which we will see in the next section.

3.5.2 Expectation conditioned on a random variable

In the previous section, we considered computing $\mathrm{E}[X]$ by breaking it up into disjoint cases $\mathrm{E}[X \mid A_1]$, $\mathrm{E}[X \mid A_2]$, etc. But keeping track of all the events in our partition of Ω is a lot of book-keeping. Conditioning on a random variable lets us combine all these conditional probabilities into a single expression

$$\mathrm{E}[X \mid Y],$$

the **expectation of X conditioned on Y**, which is defined to have the value $\mathrm{E}[X \mid Y = y]$ whenever $Y = y$.[7]

Note that $\mathrm{E}[X \mid Y]$ is generally a function of Y, unlike $\mathrm{E}[X]$ which is a constant. This also means that $\mathrm{E}[X \mid Y]$ is a random variable, and its value can depend on which outcome ω we picked from our probability space Ω. The intuition behind the definition is that $\mathrm{E}[X \mid Y]$ is the best estimate we can make of X given that we know the value of Y but nothing else.

If we want to be formal about the definition, we can specific the value of $\mathrm{E}[X \mid Y]$ explicitly for each point $\omega \in \Omega$:

$$\mathrm{E}[X \mid Y](\omega) = \mathrm{E}[X \mid Y = Y(\omega)]. \tag{3.5.4}$$

This is just another way of saying what we said already: if you want to know what the expectation of X is conditioned on Y when you get outcome ω, find the value of Y at ω and condition on seeing that.

Here is a simple example. Suppose that X and Y are independent fair coin-flips that take on the values 0 and 1 with equal probability. Then our probability space Ω has four elements, and looks like this:

$$\begin{array}{cc} \langle 0,0\rangle & \langle 0,1\rangle \\ \langle 1,0\rangle & \langle 1,1\rangle \end{array}$$

where each tuple $\langle x, y\rangle$ gives the values of X and Y.

We can also define the total number of heads as $Z = X + Y$. If we label all the points ω in our probability space with $Z(\omega)$, we get a picture that looks like this:

$$\begin{array}{cc} 0 & 1 \\ 1 & 2 \end{array}$$

This is what we see if we know the exact value of both coin-flips (or at least the exact value of Z).

[7] If Y is not discrete, the situation is more complicated. See [Fel71, §§III.2 and V.9–V.11] or [GS92, §7.9].

But now suppose we only know X, and want to compute $\mathrm{E}[Z \mid X]$. When $X = 0$, $\mathrm{E}[Z \mid X = 0] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$; and when $X = 1$, $\mathrm{E}[Z] X = 1 = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 = \frac{3}{2}$. So drawing $\mathrm{E}[Z \mid X]$ over our probability space gives

$$\begin{matrix} \frac{1}{2} & \frac{1}{2} \\ \frac{3}{2} & \frac{3}{2} \end{matrix}$$

We've averaged the value of Z across each row, since each row corresponds to one of the possible values of X.

If instead we compute $\mathrm{E}[Z \mid Y]$, we get this picture instead:

$$\begin{matrix} \frac{1}{2} & \frac{3}{2} \\ \frac{1}{2} & \frac{3}{2} \end{matrix}$$

Now instead of averaging across rows (values of X) we average across columns (values of Y). So the left column shows $\mathrm{E}[Z \mid Y = 0] = \frac{1}{2}$ and the right columns shows $\mathrm{E}[Z \mid Y = 1] = \frac{3}{2}$, which is pretty much what we'd expect.

Nothing says that we can only condition on X and Y. What happens if we condition on Z?

Now we are going to get fixed values for each possible value of Z. If we compute $\mathrm{E}[X \mid Z]$, then when $Z = 0$ this will be 0 (because $Z = 0$ implies $X = 0$), and when $Z = 2$ this will be 1 (because $Z = 2$ implies $X = 1$). The middle case is $\mathrm{E}[X \mid Z = 1] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$, because the two outcomes $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ that give $Z = 1$ are equally likely. The picture is

$$\begin{matrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{matrix}$$

3.5.2.1 Calculating conditional expectations

Usually we will not try to $\mathrm{E}[X \mid Y]$ individually for each possible ω or even each possible value y of Y. Instead, we can use various basic facts to compute $\mathrm{E}[X \mid Y]$ by applying arithmetic to random variables.

The two basic facts to start with are:

1. If X is a function of Y, then $\mathrm{E}[X \mid Y] = X$. Proof: Suppose $X = f(Y)$. From (3.5.4), for each outcome ω, we have $\mathrm{E}[X \mid Y](\omega) = \mathrm{E}[X \mid Y = Y(\omega)] = \mathrm{E}[f(Y(\omega)) \mid Y = Y(\omega)] = f(Y(\omega)) = X(\omega)$.

2. If X is independent of Y, then $\mathrm{E}[X \mid Y] = \mathrm{E}[X]$. Proof: Now for each ω, we have $\mathrm{E}[X \mid Y](\omega) = \mathrm{E}[X \mid Y = Y(\omega)] = \mathrm{E}[X]$.

We also have a rather strong version of linearity of expectation. If A and B are both functions of Z, then

$$\mathrm{E}\left[AX + BY \mid Z\right] = A\,\mathrm{E}\left[X \mid Z\right] + B\,\mathrm{E}\left[Y \mid Z\right]. \tag{3.5.5}$$

Here is a proof for discrete probability spaces. For each value z of Z, we have

$$\begin{aligned}
\mathrm{E}\left[A(Z)X + B(Z)Y \mid Z = z\right] &= \sum_{\omega \in Z^{-1}(z)} \frac{\Pr\left[\omega\right]}{\Pr\left[Z = z\right]}(A(z)X(\omega) + B(z)Y(\omega) \\
&= A(z) \sum_{\omega \in Z^{-1}(z)} \frac{\Pr\left[\omega\right]}{\Pr\left[Z = z\right]} X(\omega) \\
&\quad + B(z) \sum_{\omega \in Z^{-1}(z)} \frac{\Pr\left[\omega\right]}{\Pr\left[Z = z\right]} Y(\omega) \\
&= A(z)\,\mathrm{E}\left[X \mid Z = z\right] + B(z)\,\mathrm{E}\left[Y \mid Z = z\right],
\end{aligned}$$

which is the value when $Z = z$ of $A\,\mathrm{E}\left[X \mid Z\right] + B\,\mathrm{E}\left[Y \mid Z\right]$.

This means that we can quickly simplify many conditional expectations. If we go back to the example of the previous section, where $Z = X + Y$ is the sum of two independent fair coin-flips X and Y, then we can compute

$$\begin{aligned}
\mathrm{E}\left[Z \mid X\right] &= \mathrm{E}\left[X + Y \mid X\right] \\
&= \mathrm{E}\left[X \mid X\right] + \mathrm{E}\left[Y \mid X\right] \\
&= X + \mathrm{E}\left[Y\right] \\
&= X + \frac{1}{2}.
\end{aligned}$$

Similarly, $\mathrm{E}\left[Z \mid Y\right] = \mathrm{E}\left[X + Y \mid Y\right] = \frac{1}{2} + Y$.

In some cases we have enough additional information to run this in reverse. If we know Z and want to estimate X, we can use the fact that X and Y are symmetric to argue that $\mathrm{E}\left[X \mid Z\right] = \mathrm{E}\left[Y \mid Z\right]$. But then $Z = \mathrm{E}\left[X + Y \mid Z\right] = 2\,\mathrm{E}\left[X \mid Z\right]$, so $\mathrm{E}\left[X \mid Z\right] = Z/2$. Note that this works in general only if the events $[X = a, Y = b]$ and $[X = b, Y = a]$ have the same probabilities for all a and b even if we condition on Z, which in this case follows from the fact that X and Y are independent and identically distributed and that addition is commutative. Other cases may be messier.[8]

Other facts, like $X \geq Y$ implies $\mathrm{E}\left[X \mid Z\right] \geq \mathrm{E}\left[Y \mid Z\right]$, can be proved using similar techniques.

[8] An example with X and Y identically distributed but not independent is to imagine that we roll a six-sided die to get X, and let $Y = X + 1$ if $X < 6$ and $Y = 1$ if $X = 6$. Now knowing $Z = X + Y = 3$ tells me that $X = 1$ and $Y = 2$ exactly, neither of which is $Z/2$.

3.5.2.2 The law of iterated expectation

The **law of iterated expectation** says that

$$\mathrm{E}\left[X\right] = \mathrm{E}\left[\mathrm{E}\left[X \mid Y\right]\right]. \tag{3.5.6}$$

When Y is discrete, this is just (3.5.3) in disguise. For each of the (countably many) values of Y that occurs with nonzero probability, let A_y be the event $[Y = y]$. Then these events are a countable partition of Ω, and

$$\begin{aligned}\mathrm{E}\left[\mathrm{E}\left[X \mid Y\right]\right] &= \sum_y \Pr\left[Y = y\right] \mathrm{E}\left[\mathrm{E}\left[X \mid Y\right] \mid Y = y\right] \\ &= \sum_y \Pr\left[Y = y\right] \mathrm{E}\left[X \mid Y = y\right] \\ &= \mathrm{E}\left[X\right].\end{aligned}$$

The trick here is that we use (3.5.3) to expand out the original expression in terms of the events A_y, then notice that $\mathrm{E}\left[X \mid Y\right]$ is equal to $\mathrm{E}\left[X \mid Y = y\right]$ whenever $Y = y$.

So as claimed, conditioning on a variable gives a way to write averaging over cases very compactly.

It's also not too hard to show that iterated expectation works with partial conditioning:

$$\mathrm{E}\left[\mathrm{E}\left[X \mid Y, Z\right] \mid Y\right] = \mathrm{E}\left[X \mid Y\right]. \tag{3.5.7}$$

3.5.2.3 Conditional expectation as orthogonal projection

If you are comfortable with linear algebra, it may be helpful to think about expectation conditioned on a random variable as a form of projection onto a subspace. In this section, we'll give a brief description of how this works for a finite, discrete probability space. For a more general version, see for example [GS92, §7.9].

Consider the set of all real-valued random variables on the probability space $\{\mathsf{TT}, \mathsf{TH}, \mathsf{HT}, \mathsf{HH}\}$ corresponding to flipping two independent fair coins. We can think of each such random variable as a vector in $\mathbb{R}^4$, where the four coordinates give the value of the variable on the four possible outcomes. For example, the indicator variable for the event that the first coin is heads would look like $X = \langle 0, 0, 1, 1\rangle$ and the indicator variable for the event that the second coin is heads would look like $Y = \langle 0, 1, 0, 1\rangle$.

When we add two random variables together, we get a new random variable. This corresponds to vector addition: $X + Y = \langle 0, 0, 1, 1\rangle + \langle 0, 1, 0, 1\rangle =$

$\langle 0, 1, 1, 2\rangle$. Multiplying a random variable by a constant looks like scalar multiplication $2X = 2 \cdot \langle 0, 0, 1, 1\rangle = \langle 0, 0, 2, 2\rangle$. Because random variables support both addition and scalar multiplication, and because these operations obey the axioms of a **vector space**, we can treat the set of all real-valued random variables defined on a given probability space as a vector space, and apply all the usual tools from linear algebra to this vector space.

One thing in particular we can look at is subspaces of this vector space. Consider the set of all random variables that are functions of X. These are vectors of the form $\langle a, a, b, b\rangle$, and adding any two such vectors or multiplying any such vector by a scalar yields another vector of this form. So the functions of X form a two-dimensional subspace of the four-dimensional space of all random variables. An even lower-dimensional subspace is the one-dimensional subspace of constants: vectors of the form $\langle a, a, a, a\rangle$. As with functions of X, this set is closed under addition and multiplication by a constant.

When we take the expectation of X, we are looking for a constant that gives us the average value of X. In vector terms, this means that $\mathrm{E}[\langle 0, 0, 1, 1\rangle] = \langle 1/2, 1/2, 1/2, 1/2\rangle$. This expectation vector is in fact the orthogonal projection of $\langle 0, 0, 1, 1\rangle$ onto the subspace generated by $\mathbf{1} = \langle 1, 1, 1, 1\rangle$; we can tell this because the dot-product of $X - \mathrm{E}[X]$ with $\mathbf{1}$ is $\langle -1/2, -1/2, 1/2, 1/2\rangle \cdot \langle 1, 1, 1, 1\rangle = 0$. If instead we take a conditional expectation, we are again doing an orthogonal projection, but now onto a higher-dimensional subspace. So $\mathrm{E}[X + Y \mid X] = \langle 1/2, 1/2, 3/2, 3/2\rangle$ is the orthogonal projection of $X + Y = \langle 0, 1, 1, 2\rangle$ onto the space of all functions of X, which is generated by the basis vectors $\langle 0, 0, 1, 1\rangle$ and $\langle 1, 1, 0, 0\rangle$. As in the simple expectation, the dot-product of $(X + Y) - \mathrm{E}[X + Y \mid X]$ with either of these basis vectors is 0.

Many facts about conditional expectation translate in a straightforward way to facts about projections. Linearity of expectation is equivalent to linearity of projection onto a subspace. The law of iterated expectation $\mathrm{E}[\mathrm{E}[X \mid Y]] = \mathrm{E}[X]$ says that projecting onto the subspace of functions of Y and then onto the subspace of constants is equivalent to projection directly down to the subspace of constants; this is true in general for projection operations. It's also possible to represent other features of probability spaces in terms of expectations; for example, $\mathrm{E}[XY]$ acts like an inner product for random variables, $\mathrm{E}[X^2]$ acts like the square of the Euclidean distance, and the fact that $\mathrm{E}[X]$ is an orthogonal projection of X means that $\mathrm{E}[X]$ is precisely the constant value μ that minimizes the distance $\mathrm{E}[(X - \mu)^2]$. We won't actually use any of these facts in the following, but having another way to look at conditional expectation may be helpful in understanding how it works.

3.5.3 Expectation conditioned on a σ-algebra

Expectation conditioned on a random variable is actually a special case of the expectation of X conditioned on a *σ-algebra* $\mathcal{F}$. Recall that a σ-algebra is a family of subsets of Ω that includes Ω and is closed under complement and countable union; for discrete probability spaces, this turns out to be the set of all unions of equivalence classes for some equivalence relation on Ω,[9] and we think of $\mathcal{F}$ as representing knowledge of which equivalence class we are in, but not which point in the equivalence class we land on. An example would be if Ω consists of all values (X_1, X_2) obtained from two die rolls, and $\mathcal{F}$ consists of all sets A such that whenever one point ω with $X_1(\omega)+X_2(\omega) = s$ is in A, so is every other point ω' with $X_1(\omega') + X_2(\omega') = s$. (This is the σ-algebra **generated by** the random variable $X_1 + X_2$.)

A discrete random variable X is **measurable** with respect to $\mathcal{F}$, or **$\mathcal{F}$-measurable**, if every event $[X = x]$ is contained in $\mathcal{F}$; in other words, knowing only where we are in $\mathcal{F}$, we can compute exactly the value of X. This gives a formal way to define $\sigma(X)$: it is the smallest σ-algebra $\mathcal{F}$ such that X is $\mathcal{F}$-measurable.

If X is not $\mathcal{F}$-measurable, the best approximation we can make to it given that we only know where we are in $\mathcal{F}$ is $\mathrm{E}[X \mid \mathcal{F}]$, which is defined as a random variable Q that is (a) $\mathcal{F}$-measurable; and (b) satisfies $\mathrm{E}[Q \mid A] = \mathrm{E}[X \mid A]$ for any event $A \in \mathcal{F}$ with $\Pr[A] \neq 0$.

For discrete probability spaces, this just means that we replace X with its average value across each equivalence class: property (a) is satisfied because $\mathrm{E}[X \mid \mathcal{F}]$ is constant across each equivalence class, meaning that $[\mathrm{E}[X \mid \mathcal{F}] = x]$ is a union of equivalence classes, and property (b) is satisfied because we define $\mathrm{E}[\mathrm{E}[X \mid \mathcal{F}] \mid A] = \mathrm{E}[X \mid A]$ for each equivalence class A, and the same holds for unions of equivalence classes by a simple calculation.

This gives the same result as $\mathrm{E}[X \mid Y]$ if $\mathcal{F}$ is generated by Y, or more generally as $\mathrm{E}[X \mid Y_1, Y_2, \ldots]$ if $\mathcal{F}$ is generated by $Y_1, Y_2, \ldots$. In each case the intuition is that we are getting the best estimate we can for X given the information we have. It is also possible to define $\mathrm{E}[X \mid \mathcal{F}]$ as a projection onto the subspace of all random variables that are $\mathcal{F}$-measurable, analogously to the special case for $\mathrm{E}[X \mid Y]$ described in §3.5.2.3.

[9] Proof: Let $\mathcal{F}$ be a σ-algebra over a countable set Ω. Let $\omega \sim \omega'$ if, for all A in $\mathcal{F}$, $\omega \in A$ if and only if $\omega' \in A$; this is an equivalence relation on Ω. To show that the equivalence classes of $\sim$ are elements of $\mathcal{F}$, for each $\omega'' \not\sim \omega$, let $A_{\omega''}$ be some element of $\mathcal{F}$ that contains ω but not ω''. Then $\bigcap_{\omega''} A_{\omega''}$ (a countable intersection of elements of $\mathcal{F}$) contains ω and all points $\omega' \sim \omega$ but no points $\omega'' \not\sim \omega$; in other words, it's the equivalence class of ω. Since there are only countably many such equivalence classes, we can construct all the elements of $\mathcal{F}$ by taking all possible unions of them.

Sometimes it is convenient to use more than one σ-algebra to represent increasing knowledge over time. A **filtration** is a sequence of σ-algebras $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$, where each $\mathcal{F}_t$ represents the information we have available at time t. That each $\mathcal{F}_t$ is a subset of $\mathcal{F}_{t+1}$ means that any event we can determine at time t we can also determine at all future times $t' > t$: though we may learn more information over time, we never forget what we already know. A common example of a filtration is when we have a sequence of random variables $X_1, X_2, \ldots$, and define $\mathcal{F}_t$ as the σ-algebra $\langle X_1, X_2, \ldots, X_t \rangle$ generated by $X_1, X_2, \ldots, X_t$.

When one σ-algebra is a subset of another, a version of the law of iterated expectation applies: $\mathcal{F} \in \mathcal{F}'$ implies $\mathrm{E}\left[\mathrm{E}\left[X \mid \mathcal{F}'\right] \mid \mathcal{F}\right] = \mathrm{E}\left[X \mid \mathcal{F}\right]$. One way to think about this is that if we forget everything about X we can't predict from $\mathcal{F}'$ and then forget everything that's left that we can't predict from $\mathcal{F}$, we get to the same place as if we just forget everything except $\mathcal{F}$ to begin with. The simplest version $\mathrm{E}\left[X \mid \mathcal{F}'\right] = \mathrm{E}\left[X\right]$ is just what happens when $\mathcal{F}$ is the trivial σ-algebra $\{\emptyset, \Omega\}$, where all we know is that something happened, but we don't know what.

3.5.4 Examples

- Let X be the value of a six-sided die. Let A be the event that X is even. Then

$$\begin{aligned}\mathrm{E}\left[X \mid A\right] &= \sum_x x \Pr\left[X = x \mid A\right] \\ &= (2 + 4 + 6) \cdot \frac{1}{3} \\ &= 4.\end{aligned}$$

- Let X and Y be independent six-sided dice, and let $Z = X + Y$. Then $\mathrm{E}\left[Z \mid X\right]$ is a random variable whose value is $1 + 7/2$ when $X = 1$, $2 + 7/2$ when $X = 2$, etc. We can write this succinctly by writing $\mathrm{E}\left[Z \mid X\right] = X + 7/2$.

- Conversely, if X, Y, and Z are as above, we can also compute $\mathrm{E}\left[X\right] Z$. Here we are told what Z is and must make an estimate of X.

  For some values of Z, this nails down X completely: $\mathrm{E}\left[X \mid Z = 2\right] = 1$ because X you can only make 2 in this model as $1+1$. For other values, we don't know much about X, but can still compute the expectation. For example, to compute $\mathrm{E}\left[X \mid Z = 5\right]$, we have to average X over all the pairs (X, Y) that sum to 5. This gives $\mathrm{E}\left[X \mid Z = 5\right] = \frac{1}{4}(1+2+3+$

4) = $\frac{5}{2}$. (This is not terribly surprising, since by symmetry $\mathrm{E}[Y \mid Z = 5]$ should equal $\mathrm{E}[X \mid Z = 5]$, and since conditional expectations add just like regular expectations, we'd expect that the sum of these two expectations would be 5.)

The actual random variable $\mathrm{E}[X \mid Z]$ summarizes these conditional expectations for all events of the form $[Z = z]$. Because of the symmetry argument above, we can write it succinctly as $\mathrm{E}[X \mid Z] = \frac{Z}{2}$. Or we could list its value for every ω in our underlying probability space, as in done in Table 3.2 for this and various other conditional expectations on the two-independent-dice space.

3.6 Applications

3.6.1 Yao's lemma

In Section 1.1, we considered a special case of the unordered search problem, where we have an unordered array $A[1..n]$ and want to find the location of a specific element x. For deterministic algorithms, this requires probing n array locations in the worst case, because the adversary can place x in the last place we look. Using a randomized algorithm, we can reduce this to $(n + 1)/2$ probes on average, either by probing according to a uniform random permutation or just by probing from left-to-right or right-to-left with equal probability.

Can we do better? Proving lower bounds is a nuisance even for deterministic algorithms, and for randomized algorithms we have even more to keep track of. But there is a sneaky trick that allows us to reduce randomized lower bounds to deterministic lower bounds in many cases.

The idea is that if we have a randomized algorithm that runs in time $T(x, r)$ on input x with random bits r, then for any fixed choice of r we have a deterministic algorithm. So for each n, we find some random X with $|X| = n$ and show that, for any deterministic algorithm that runs in time $T'(x)$, $\mathrm{E}[T'(X)] \geq f(n)$. But then $\mathrm{E}[T(X, R)] = \mathrm{E}[\mathrm{E}[T(X, R) \mid R]] = \mathrm{E}[\mathrm{E}[T_R(X) \mid R]] \geq f(n)$.

This gives us **Yao's lemma**:

Lemma 3.6.1 (Yao's lemma (informal version)[Yao77])**.** *Fix some problem. Suppose there is a random distribution on inputs X of size n such that every deterministic algorithm for the problem has expected cost $T(n)$.*

Then the worst-case expected cost of any randomized algorithm is at least $T(n)$.

ω	X	Y	$Z = X + Y$	$\mathrm{E}[X]$	$\mathrm{E}[X \mid Y = 3]$	$\mathrm{E}[X \mid Y]$	$\mathrm{E}[Z \mid X]$	$\mathrm{E}[X \mid Z]$	$\mathrm{E}[X \mid X]$
$(1,1)$	1	1	2	7/2	7/2	7/2	$1 + 7/2$	2/2	1
$(1,2)$	1	2	3	7/2	7/2	7/2	$1 + 7/2$	3/2	1
$(1,3)$	1	3	4	7/2	7/2	7/2	$1 + 7/2$	4/2	1
$(1,4)$	1	4	5	7/2	7/2	7/2	$1 + 7/2$	5/2	1
$(1,5)$	1	5	6	7/2	7/2	7/2	$1 + 7/2$	6/2	1
$(1,6)$	1	6	7	7/2	7/2	7/2	$1 + 7/2$	7/2	1
$(2,1)$	2	1	3	7/2	7/2	7/2	$2 + 7/2$	3/2	2
$(2,2)$	2	2	4	7/2	7/2	7/2	$2 + 7/2$	4/2	2
$(2,3)$	2	3	5	7/2	7/2	7/2	$2 + 7/2$	5/2	2
$(2,4)$	2	4	6	7/2	7/2	7/2	$2 + 7/2$	6/2	2
$(2,5)$	2	5	7	7/2	7/2	7/2	$2 + 7/2$	7/2	2
$(2,6)$	2	6	8	7/2	7/2	7/2	$2 + 7/2$	8/2	2
$(3,1)$	3	1	4	7/2	7/2	7/2	$3 + 7/2$	4/2	3
$(3,2)$	3	2	5	7/2	7/2	7/2	$3 + 7/2$	5/2	3
$(3,3)$	3	3	6	7/2	7/2	7/2	$3 + 7/2$	6/2	3
$(3,4)$	3	4	7	7/2	7/2	7/2	$3 + 7/2$	7/2	3
$(3,5)$	3	5	8	7/2	7/2	7/2	$3 + 7/2$	8/2	3
$(3,6)$	3	6	9	7/2	7/2	7/2	$3 + 7/2$	9/2	3
$(4,1)$	4	1	5	7/2	7/2	7/2	$4 + 7/2$	5/2	4
$(4,2)$	4	2	6	7/2	7/2	7/2	$4 + 7/2$	6/2	4
$(4,3)$	4	3	7	7/2	7/2	7/2	$4 + 7/2$	7/2	4
$(4,4)$	4	4	8	7/2	7/2	7/2	$4 + 7/2$	8/2	4
$(4,5)$	4	5	9	7/2	7/2	7/2	$4 + 7/2$	9/2	4
$(4,6)$	4	6	10	7/2	7/2	7/2	$4 + 7/2$	10/2	4
$(5,1)$	5	1	6	7/2	7/2	7/2	$5 + 7/2$	6/2	5
$(5,2)$	5	2	7	7/2	7/2	7/2	$5 + 7/2$	7/2	5
$(5,3)$	5	3	8	7/2	7/2	7/2	$5 + 7/2$	8/2	5
$(5,4)$	5	4	9	7/2	7/2	7/2	$5 + 7/2$	9/2	5
$(5,5)$	5	5	10	7/2	7/2	7/2	$5 + 7/2$	10/2	5
$(5,6)$	5	6	11	7/2	7/2	7/2	$5 + 7/2$	11/2	5
$(6,1)$	6	1	7	7/2	7/2	7/2	$6 + 7/2$	7/2	6
$(6,2)$	6	2	8	7/2	7/2	7/2	$6 + 7/2$	8/2	6
$(6,3)$	6	3	9	7/2	7/2	7/2	$6 + 7/2$	9/2	6
$(6,4)$	6	4	10	7/2	7/2	7/2	$6 + 7/2$	10/2	6
$(6,5)$	6	5	11	7/2	7/2	7/2	$6 + 7/2$	11/2	6
$(6,6)$	6	6	12	7/2	7/2	7/2	$6 + 7/2$	12/2	6

Table 3.2: Various conditional expectations on two independent dice

For unordered search, putting x in a uniform random array location makes any deterministic algorithm take at least $(n+1)/2$ probes on average. So randomized algorithms take at least $(n+1)/2$ probes as well.

3.6.2 Geometric random variables

Suppose that we are running a Las Vegas algorithm that takes a fixed amount of time T, but succeeds only with probability p (which we take to be independent of the outcome of any other run of the algorithm). If the algorithm fails, we run it again. How long does it take on average to get the algorithm to work?

We can reduce the problem to computing $\mathrm{E}[TX] = T\,\mathrm{E}[X]$, where X is the number of times the algorithm runs. The probability that $X = n$ is exactly $(1-p)^{n-1}p$, because we need to get $n-1$ failures with probability $1-p$ each followed by a single success with probability p, and by assumption all of these probabilities are independent. A variable with this kind of distribution is called a **geometric random variable**. We saw a special case of this distribution earlier (§2.3.3.1) when we were looking at how many flips it would take on average to get the first tails from a fair coin coin (in that case, p was $1/2$).

Using conditional expectation, it's straightforward to compute $\mathrm{E}[X]$. Let A be the event that the algorithm succeeds on the first run, i.e., then event $[X = 1]$. Then

$$\begin{aligned}\mathrm{E}[X] &= \mathrm{E}[X \mid A]\Pr[A] + \mathrm{E}\left[X \;\middle|\; \bar{A}\right]\Pr\left[\bar{A}\right] \\ &= 1 \cdot p + \mathrm{E}\left[X \;\middle|\; \bar{A}\right] \cdot (1-p).\end{aligned}$$

The tricky part here is to evaluate $\mathrm{E}\left[X \;\middle|\; \bar{A}\right]$. Intuitively, if we don't succeed the first time, we've wasted one step and are back where we started, so it should be the case that $\mathrm{E}\left[X \;\middle|\; \bar{A}\right] = 1 + \mathrm{E}[X]$. If we want to be really careful,

we can calculate this out formally (no sensible person would ever do this):

$$\begin{aligned}
\mathrm{E}\left[X \;\middle|\; \bar{A}\right] &= \sum_{n=1}^{\infty} n \Pr\left[X = n \mid X \neq 1\right] \\
&= \sum_{n=2}^{\infty} n \frac{\Pr\left[X = n\right]}{\Pr\left[X \neq 1\right]} \\
&= \sum_{n=2}^{\infty} n \frac{(1-p)^{n-1}p}{1-p} \\
&= \sum_{n=2}^{\infty} n(1-p)^{n-2}p \\
&= \sum_{n=1}^{\infty} (n+1)(1-p)^{n-1}p \\
&= 1 + \sum_{n=1}^{\infty} n(1-p)^{n-1}p \\
&= 1 + \mathrm{E}\left[X\right].
\end{aligned}$$

Since we know that $\mathrm{E}\left[X\right] = p + (1 + \mathrm{E}\left[X\right])(1-p)$, a bit of algebra gives $\mathrm{E}\left[X\right] = 1/p$, which is about what we'd expect.

There are more direct ways to get the same result. If we don't have conditional expectation to work with, we can try computing the sum $\mathrm{E}\left[X\right] = \sum_{n=1}^{\infty} n(1-p)^{n-1}p$ directly. The easiest way to do this is probably to use generating functions (see, for example, [GKP88, Chapter 7] or [Wil06]). An alternative argument is given in [MU17, §2.4]; this uses the fact that $\mathrm{E}\left[X\right] = \sum_{n=1}^{\infty} \Pr\left[X \geq n\right]$, which holds when X takes on only non-negative integer values.

3.6.3 Coupon collector

In the **coupon collector problem**, we throw balls uniformly and independently into n bins until every bin has at least one ball. When this happens, how many balls have we used on average?[10]

Let X_i be the number of balls needed to go from $i-1$ nonempty bins to i nonempty bins. It's easy to see that $X_1 = 1$ always. For larger i, each time we throw a ball, it lands in an empty bin with probability $\frac{n-i+1}{n}$. This

[10]The name comes from the problem of collecting coupons at random until you have all of them. A typical algorithmic application is having a cluster of machines choose jobs to finish at random from some list until all are done. The expected number of job executions to complete n jobs is given exactly by the solution to the coupon collector problem.

means that X_i has a geometric distribution with probability $\frac{n-i+1}{n}$, giving $\mathrm{E}[X_i] = \frac{n}{n-i+1}$ from the analysis in §3.6.2.

To get the total expected number of balls, take the sum

$$\begin{aligned}\mathrm{E}\left[\sum_{i=1}^{n} X_i\right] &= \sum_{i=1}^{n} \mathrm{E}[X_i] \\ &= \sum_{i=1}^{n} \frac{n}{n-i+1} \\ &= n\sum_{i=1}^{n} \frac{1}{i} \\ &= nH_n.\end{aligned}$$

In asymptotic terms, this is $\Theta(n \log n)$.

3.6.4 Hoare's FIND

Hoare's FIND [Hoa61b], often called **QuickSelect**, is an algorithm for finding the k-th smallest element of an unsorted array that works like QuickSort, only after partitioning the array around a random pivot we throw away the part that doesn't contain our target and recurse only on the surviving piece. As with QuickSort, we'd like to compute the expected number of comparisons used by this algorithm, on the assumption that the cost of the comparisons dominates the rest of the costs of the algorithm.

Here the indicator-variable trick gets painful fast. It turns out to be easier to get an upper bound by computing the expected number of elements that are left after each split.

First, let's analyze the pivot step. If the pivot is chosen uniformly, the number of elements X smaller than the pivot is uniformly distributed in the range 0 to $n-1$. The number of elements larger than the pivot will be $n-X-1$. In the worst case, we find ourselves recursing on the large pile always, giving a bound on the number of survivors Y of $Y \le \max(X, n-X+1)$.

What is the expected value of Y? By considering both ways the max can go, we get

$$\begin{aligned}\mathrm{E}[Y] = {} & \mathrm{E}[X \mid X > n-X+1]\Pr[X > n-X+1] \\ & + \mathrm{E}[n-X+1 \mid n-X+1 \ge X]\Pr[n-X+1 \ge X].\end{aligned}$$

For both conditional expectations, we are choosing a value uniformly in either the range $\lceil \frac{n-1}{2} \rceil$ to $n-1$ or $\lceil \frac{n-1}{2} \rceil + 1$ to $n-1$, and in either case the

expectation will be equal to the average of the two endpoints by symmetry. So we get

$$\begin{aligned}\mathrm{E}[Y] &\leq \frac{n/2+n-1}{2}\Pr[X > n-X+1] + \frac{n/2+n}{2}\Pr[n-X+1 \geq X] \\ &= \left(\frac{3}{4}n - \frac{1}{2}\right)\Pr[X > n-X+1] + \frac{3}{4}n\Pr[n-X+1 \geq X] \\ &\leq \frac{3}{4}n.\end{aligned}$$

Now let X_i be the number of survivors after i pivot steps. Note that $\max(0, X_i - 1)$ gives the number of comparisons at the following pivot step, so that $\sum_{i=0}^{\infty} X_i$ is an upper bound on the number of comparisons.

We have $X_0 = n$, and from the preceding argument $\mathrm{E}[X_1] \leq (3/4)n$. But more generally, we can use the same argument to show that $\mathrm{E}[X_{i+1} \mid X_i] \leq (3/4)X_i$, and by induction $\mathrm{E}[X_i] \leq (3/4)^i n$. We also have that $X_j = 0$ for all $j \geq n$, because we lose at least one element (the pivot) at each pivoting step. This saves us from having to deal with an infinite sum.

Using linearity of expectation,

$$\begin{aligned}\mathrm{E}\left[\sum_{i=0}^{\infty} X_i\right] &= \mathrm{E}\left[\sum_{i=0}^{n} X_i\right] \\ &= \sum_{i=0}^{n} \mathrm{E}[X_i] \\ &\leq \sum_{i=0}^{n} (3/4)^i n \\ &\leq 4n.\end{aligned}$$

Chapter 4

Basic probabilistic inequalities

Here we're going to look at some inequalities useful for proving properties of randomized algorithms. These come in two flavors: inequalities involving probabilities, which are useful for bounding the probability that something bad happens, and inequalities involving expectations, which are used to bound expected running times. Later, in Chapter 5, we'll be doing both, by looking at inequalities that show that a random variable is close to its expectation with high probability.[1]

4.1 Markov's inequality

This is the key tool for turning expectations of non-negative random variables into (upper) bounds on probabilities. Used directly, it generally doesn't give very good bounds, but it can work well if we apply it to $\mathrm{E}[f(X)]$ for a fast-growing function f; for some examples, see Chebyshev's inequality (§5.1) or Chernoff bounds (§5.2).

[1] Often, the phrase **with high probability** is used in algorithm analysis to mean specifically with probability at least $1 - n^{-c}$ for any fixed c. The idea is that if an algorithm works with high probability in this sense, then the probability that it fails each time you run it is at most n^{-c}, which means that if you run it as a subroutine in a polynomial-time algorithm that calls is at most $n^{c'}$ times, the total probability of failure is at most $n^{c'} n^{-c} = n^{c'-c}$ by the union bound. Assuming we can pick c to be much larger than c', this makes the outer algorithm also work with high probability.

Markov's inequality says that if $X \geq 0$ and $\alpha > 0$, then

$$\Pr[X \geq \alpha] \leq \frac{\mathrm{E}[X]}{\alpha}.$$

The proof is immediate from the law of total probability (2.3.3). We have

$$\begin{aligned}\mathrm{E}[X] &= \mathrm{E}[X \mid X \geq \alpha] \Pr[X \geq \alpha] + \mathrm{E}[X \mid x < \alpha] \Pr[X < \alpha] \\ &\geq \alpha \cdot \Pr[X \geq \alpha] + 0 \cdot \Pr[X < \alpha] \\ &= \alpha \cdot \Pr[X \geq \alpha];\end{aligned}$$

now solve for $\Pr[X \geq \alpha]$.

Markov's inequality doesn't work in reverse. For example, consider the following game: for each integer $k > 0$, with probability 2^{-k}, I give you 2^k dollars. Letting X be your payoff from the game, we have $\Pr\left[X \geq 2^k\right] = \sum_{j=k}^{\infty} 2^{-k} = 2^{-k} \sum_{\ell=0}^{\infty} 2^{-\ell} = \frac{2}{2^k}$. The right-hand side here is exactly what we would get from Markov's inequality if $\mathrm{E}[X] = 2$. But in this case, $\mathrm{E}[X] \neq 2$; in fact, the expectation of X is given by $\sum_{k=1}^{\infty} 2^k 2^{-k}$, which diverges.

4.1.1 Applications

4.1.1.1 Sum of fair coins

Flip n independent fair coins, and let S be the number of heads we get. Since $\mathrm{E}[S] = n/2$, we get $\Pr[S = n] = 1/2$. This is much larger than the actual value 2^{-n}, but it's the best we can hope for if we only know $\mathrm{E}[S]$: if we let S be 0 or n with equal probability, we also get $\mathrm{E}[S] = n/2$.

4.1.1.2 Randomized QuickSort

The expected running time for randomized QuickSort is $O(n \log n)$. It follows that the probability that randomized QuickSort takes more than $f(n)$ time is $O(n \log n / f(n))$. For example, the probability that it performs the maximum $\binom{n}{2} = O(n^2)$ comparisons is $O(\log n / n)$. (It's possible to do much better than this.)

4.1.1.3 Balls in bins

Suppose we toss m balls in n bins, uniformly and independently. What is the probability that some particular bin contains at least k balls? The probability that a particular ball lands in a particular bin is $1/n$, so the expected number of balls in the bin is m/n. This gives a bound of m/nk

that a particular bin contains k or more balls. Unfortunately this is not a very good bound.

4.2 Union bound (Boole's inequality)

The **union bound** or **Boole's inequality** says that for any countable collection of events $\{A_i\}$,

$$\Pr\left[\bigcup A_i\right] \leq \sum \Pr\left[A_i\right]. \tag{4.2.1}$$

Combining Markov's inequality with linearity of expectation and indicator variables gives a succinct proof of the union bound:

$$\begin{aligned}\Pr\left[\bigcup A_i\right] &= \Pr\left[\sum 1_{A_i} \geq 1\right] \\ &\leq \mathrm{E}\left[\sum 1_{A_i}\right] \\ &= \sum \mathrm{E}\left[1_{A_i}\right] \\ &= \sum \Pr\left[A_i\right].\end{aligned}$$

Note that for this to work for infinitely many events we need to use the fact that 1_{A_i} is non-negative.

If we prefer to avoid any issues with infinite sums of expectations, the direct way to prove this is to replace A_i with $B_i = A_i \setminus \bigcup_{j=1}^{i-1} A_i$. Then $\bigcup A_i = \bigcup B_i$, but since the B_i are disjoint and each B_i is a subset of the corresponding A_i, we get $\Pr\left[\bigcup A_i\right] = \Pr\left[\bigcup B_i\right] = \sum \Pr\left[B_i\right] \leq \sum \Pr\left[A_i\right]$.

The typical use of the union bound is to show that if an algorithm can fail only if various improbable events occur, then the probability of failure is no greater than the sum of the probabilities of these events. This reduces the problem of showing that an algorithm works with probability $1 - \epsilon$ to constructing an **error budget** that divides the ϵ probability of failure among all the bad outcomes.

4.2.1 Example: Balls in bins

Suppose we toss n balls uniformly and independently into n bins. What high-probability bound can we get on the maximum number of balls in any one bin?[2]

[2] Algorithmic version: we insert n elements into a hash table with n positions using a random hash function. What is the maximum number of elements in any one position?

Consider all $\binom{n}{k}$ sets S of k balls. If we get at least k balls in some bin, then one of these sets must all land in the same bin. Call the event that all balls in S choose the same bin A_S. The probability that A_S occurs is exactly n^{-k+1}.

Using the union bound, we get

$$\begin{aligned}\Pr\left[\text{some bin gets at least } k \text{ balls}\right] &= \Pr\left[\bigcup_S A_S\right] \\ &\leq \sum_S \Pr\left[A_S\right] \\ &= \binom{n}{k} n^{-k+1} \\ &\leq \frac{n^k}{k!} n^{-k+1} \\ &= \frac{n}{k!}.\end{aligned}$$

If we want this probability to be low, we should choose k so that $k! \gg n$. Stirling's formula says that $k! \geq \sqrt{2\pi k}(k/e)^k \geq (k/e)^k$, which gives $\ln(k!) \geq k(\ln k - 1)$. If we set $k = c \ln n / \ln \ln n$, we get

$$\begin{aligned}\ln(k!) &\geq \frac{c \ln n}{\ln \ln n} \left(\ln c + \ln \ln n - \ln \ln \ln n - 1\right) \\ &\geq c \ln n.\end{aligned}$$

when n is sufficiently large.

It follows that the bound $n/k!$ in this case is less than $n/\exp(c \ln n) = n \cdot n^{-c} = n^{1-c}$. For suitable choice of c we get a high probability that every bin gets at most $O(\log n / \log \log n)$ balls.

4.3 Jensen's inequality

This is mostly useful if we can calculate $\mathrm{E}\left[X\right]$ easily for some X, but what we really care about is some other random variable $Y = f(X)$.

Jensen's inequality applies when f is a **convex function**, which means that for any x, y, and $0 \leq \mu \leq 1$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$. Geometrically, this means that the line segment between any two points on the graph of f never goes below f; i.e., that the set of points $\{(x, y) \mid y \geq f(x)\}$ is convex. If we want to show that a continuous function f is convex, it's enough to show that that $f\left(\frac{x+y}{2}\right) \leq \frac{f(x)+f(y)}{2}$ for all x and y (in effect, we

only need to prove it for the case $\lambda = 1/2$). If f is twice-differentiable, an even easier way is to show that $f''(x) \geq 0$ for all x.

The inequality says that if X is a random variable and f is convex then

$$f(\mathrm{E}[X]) \leq \mathrm{E}[f(X)]. \tag{4.3.1}$$

Alternatively, if f is **concave** (which means that $f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y)$, or equivalently that $-f$ is convex), the reverse inequality holds:

$$f(\mathrm{E}[X]) \geq \mathrm{E}[f(X)]. \tag{4.3.2}$$

In both cases, the direction of Jensen's inequality matches the direction of the inequality in the definition of convexity or concavity. This is not surprising because convexity or or concavity is just Jensen's inequality for the random variable X that equals x with probability λ and y with probability $1-\lambda$. Jensen's inequality just says that this continues to work for any random variable for which the expectations exist.

4.3.1 Proof

Here is a proof for the case that f is convex and differentiable. The idea is that if f is convex, then it lies above the tangent line at $\mathrm{E}[X]$. So we can define a linear function g that represents this tangent line, and get, for all x:

$$f(x) \geq g(x) = f(\mathrm{E}[X]) + (x - \mathrm{E}[X])f'(\mathrm{E}[X]). \tag{4.3.3}$$

But then

$$\begin{aligned}
\mathrm{E}[f(X)] &\geq \mathrm{E}[g(X)] \\
&= \mathrm{E}[f(\mathrm{E}[X]) + (X - \mathrm{E}[X])f'(\mathrm{E}[X])] \\
&= f(\mathrm{E}[X]) + \mathrm{E}[(X - \mathrm{E}[X])]\, f'(\mathrm{E}[X]) \\
&= f(\mathrm{E}[X]) + (\mathrm{E}[X] - \mathrm{E}[X])f'(\mathrm{E}[X]) \\
&= f(\mathrm{E}[X]).
\end{aligned}$$

Figure 4.1 shows what this looks like for a particular convex f.

This is pretty much all linearity of expectation in action: $\mathrm{E}[X]$, $f(\mathrm{E}[X])$, and $f'(\mathrm{E}[X])$ are all constants, so we can pull them out of expectations whenever we see them.

The proof of the general case is similar, but for a non-differentiable convex function it takes a bit more work to show that the bounding linear function g exists.

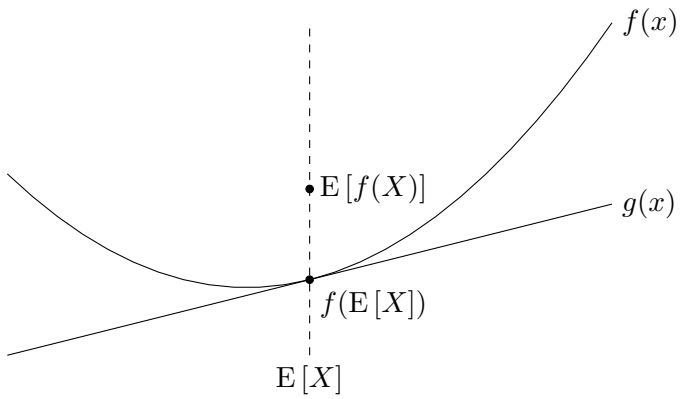


Figure 4.1: Proof of Jensen's inequality

4.3.2 Applications

4.3.2.1 Fair coins: lower bound

Suppose we flip n fair coins, and we want to get a lower bound on $\mathrm{E}\left[X^2\right]$, where X is the number of heads. The function $f : x \mapsto x^2$ is convex (take its second derivative), so (4.3.1) gives $\mathrm{E}\left[X^2\right] \geq (\mathrm{E}[X])^2 = \frac{n^2}{4}$.

The actual value for $\mathrm{E}\left[X^2\right]$ is $\frac{n^2}{4} + \frac{n}{4}$, which can be found directly using generating functions[3] or less directly using variance, which we will encounter in §5.1. This is pretty close to the lower bound we got out of Jensen's inequality, but we can't count on this happening in general.

4.3.2.2 Fair coins: upper bound

For an upper bound, we can choose a concave f. For example, if X is as in the previous example, $\mathrm{E}[\lg X] \leq \lg \mathrm{E}[X] = \lg \frac{n}{2} = \lg n - 1$. This is probably pretty close to the exact value, as we will see later that X will almost always be within a factor of $1 + o(1)$ of $n/2$. It's not a terribly useful upper bound, because if we use it with (say) Markov's inequality, the most we can prove is that $\Pr[X = n] = \Pr[\lg X = \lg n] \leq \frac{\lg n - 1}{\lg n} = 1 - \frac{1}{\lg n}$, which is an even

[3] Here's how: The **probability generating function** for X is $F(z) = \mathrm{E}\left[z^k\right] = \sum z^k \Pr[X = k] = 2^{-n}(1+z)^n$. Then $zF'(z) = 2^{-n}nz(1+z)^{n-1} = \sum_k kz^k \Pr[X = k]$. Taking the derivative of this a second time gives $2^{-n}n(1+z)^{n-1} + 2^{-n}n(n-1)z(1+z)^{n-2} = \sum_k k^2 z^{k-1} \Pr[X = k]$. Evaluate this monstrosity at $z = 1$ to get $\mathrm{E}\left[X^2\right] = \sum_k k^2 \Pr[X = k] = 2^{-n}n2^{n-1} + 2^{-n}n(n-1)2^{n-2} = n/2 + n(n-1)/4 = \frac{2n+n^2-n}{4} = n^2/4 + n/4$.

worse bound than the 1/2 we can get from applying Markov's inequality to X directly.

4.3.2.3 Sifters

Here's an example of Jensen's inequality in action in the analysis of an actual distributed algorithm. For some problems in distributed computing, it's useful to reduce coordinating a large number of processes to coordinating a smaller number. A **sifter** [AA11] is a randomized mechanism for an asynchronous shared-memory system that sorts the processes into "winners" and "losers," guaranteeing that there is at least one winner. The goal is to make the expected number of winners as small as possible. The problem is tricky, because processes can only communicate by reading and writing shared variables, and an adversary gets to choose which processes participate and fix the schedule of when each of these processes perform their operations.

The current best known sifter is due to Giakkoupis and Woelfel [GW12]. For n processes, it uses an array A of $\lceil \lg n \rceil$ bits, each of which can be read or written by any of the processes. When a process executes the sifter, it chooses a random index $r \in 1 \dots \lceil \lg n \rceil$ with probability 2^{-r-1} (this doesn't exactly sum to 1, so the excess probability gets added to $r = \lceil \lg n \rceil$). The process then writes a 1 to $A[r]$ and reads $A[r+1]$. If it sees a 0 in its read (or chooses $r = \lceil \lg n \rceil$), it wins; otherwise it loses.

This works as a sifter, because no matter how many processes participate, some process chooses a value of r at least as large as any other process's value, and this process wins. To bound the expected number of winners, take the sum over all r over the random variable W_r representing the winners who chose this particular value r. A process that chooses r wins if it carries out its read operation before any process writes $r+1$. If the adversary wants to maximize the number of winners, it should let each process read as soon as possible; this effectively means that a process that choose r wins if no process previously chooses $r+1$. Since r is twice as likely to be chosen as $r+1$, conditioning on a process picking r or $r+1$, there is only a 1/3 chance that it chooses $r+1$. So at most $1/(1/3) - 1 = 2 = O(1)$ process on average choose r before some process chooses $r+1$. (A simpler argument shows that the expected number of processes that win because they choose $r = \lceil \lg n \rceil$ is at most 2 as well.)

Summing $W_r \leq 2$ over all r gives at most $2\lceil \lg n \rceil$ winners on average. Furthermore, if $k < n$ processes participate, essentially the same analysis shows that only $2\lceil \lg k \rceil$ processes win on average. So this is a pretty effective tool for getting rid of excess processes.

But it gets better. Suppose that we take the winners of one sifter and feed them into a second sifter. Let X_k be the number of processes left after k sifters. We have that $X_0 = n$ and $\mathrm{E}[X_1] \leq 2\lceil \lg n \rceil$, but what can we say about $\mathrm{E}[X_2]$? We can calculate $\mathrm{E}[X_2] = \mathrm{E}[\mathrm{E}[X_2 \mid X_1]] \leq 2\lceil \lg X_1 \rceil$. Unfortunately, the ceiling means that $2\lceil \lg x \rceil$ is not a concave function, but $f(x) = 2(\lg x + 1) \geq 2\lceil \lg x \rceil$ is. So $\mathrm{E}[X_2] \leq f(f(n))$, and in general $\mathrm{E}[X_i] \leq f^{(i)}(n)$, where $f^{(i)}$ is the i-fold composition of f. All the extra constants obscure what is going on a bit, but with a little bit of algebra it is not too hard to show that $f^{(i)}(n) = O(1)$ for $i = O(\log^* n)$.[4] So this gets rid of all but a constant number of processes very quickly.

[4]The $\log^*$ function counts how many times you need to hit n with lg to reduce it to one or less. So $\log^* 1 = 0, \log^* 2 = 1, \log^* 4 = 2, \log^* 16 = 3, \log^* 65536 = 4, \log^* 2^{65536} = 5$, and after that it starts getting silly.

Chapter 5

Concentration bounds

If we really want to get tight bounds on a random variable X, the trick will turn out to be picking some non-negative function $f(X)$ where (a) we can calculate $\mathrm{E}[f(X)]$, and (b) f grows fast enough that merely large values of X produce huge values of $f(X)$, allowing us to get small probability bounds by applying Markov's inequality to $f(X)$. This approach is often used to show that X lies close to $\mathrm{E}[X]$ with reasonably high probability, what is known as a **concentration bound**.

Typically concentration bounds are applied to sums of random variables, which may or may not be fully independent. Which bound you may want to use often depends on the structure of your sum. A quick summary of the bounds in this chapter is given in Table 5.1. The rule of thumb is to use Chernoff bounds (§5.2) if you have a sum of independent 0–1 random variables; the Azuma-Hoeffding inequality (§5.3) if you have bounded variables with a more complicated distribution that may be less independent; and Chebyshev's inequality (§5.1) if nothing else works but you can somehow compute the variance of your sum (e.g., if the X_i are independent or have easily computed covariance). In the case of Chernoff bounds, you will almost always end up using one of the weaker but cleaner versions in §5.2.2 rather than the general version in §5.2.1.

If none of these bounds work for your particular application, there are many more out there. See for example the textbook by Dubhashi and Panconesi [DP09].

Chernoff	$X_i \in \{0,1\}$, independent	$\Pr[S \geq (1+\delta)\operatorname{E}[S]] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\operatorname{E}[S]}$
Azuma-Hoeffding	$\lvert X_i\rvert \leq c_i$, martingale	$\Pr[S \geq t] \leq \exp\left(-\frac{t^2}{2\sum c_i^2}\right)$
Chebyshev		$\Pr[\lvert S - \operatorname{E}[S]\rvert \geq \alpha] \leq \frac{\operatorname{Var}[S]}{\alpha^2}$

Table 5.1: Concentration bounds for $S = \sum X_i$ (strongest to weakest)

5.1 Chebyshev's inequality

Chebyshev's inequality allows us to show that a random variable is close to its mean, by applying Markov's inequality to the **variance** of X, defined as $\operatorname{Var}[X] = \operatorname{E}\left[(X - \operatorname{E}[X])^2\right]$. It's a fairly weak concentration bound, that is most useful when X is a sum of random variables with limited independence.

Using Markov's inequality, calculate

$$\begin{aligned}\Pr[|X - \operatorname{E}[X]| \geq \alpha] &= \Pr\left[(X - \operatorname{E}[X])^2 \geq \alpha^2\right] \\ &\leq \frac{\operatorname{E}\left[(X - \operatorname{E}[X])^2\right]}{\alpha^2} \\ &= \frac{\operatorname{Var}[X]}{\alpha^2}. \end{aligned} \tag{5.1.1}$$

5.1.1 Computing variance

At this point it would be reasonable to ask why we are going through $\operatorname{Var}[X] = \operatorname{E}\left[(X - \operatorname{E}[X])^2\right]$ rather than just using $\operatorname{E}[|X - \operatorname{E}[X]|]$. The reason is that $\operatorname{Var}[X]$ is usually easier to compute, especially if X is a sum. In this section, we'll give some examples of computing variance, including for various standard random variables that come up often in randomized algorithms.

5.1.1.1 Alternative formula

The first step is to give an alternative formula for the variance that is more convenient in some cases.

Expand

$$\begin{aligned}\mathrm{E}\left[(X - \mathrm{E}[X])^2\right] &= \mathrm{E}\left[X^2 - 2X \cdot \mathrm{E}[X] + (\mathrm{E}[X])^2\right] \\ &= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X] \cdot \mathrm{E}[X] + (\mathrm{E}[X])^2 \\ &= \mathrm{E}\left[X^2\right] - (\mathrm{E}[X])^2 . \end{aligned} \tag{5.1.2}$$

This formula is easier to use if you are estimating the variance from a sequence of samples; by tracking $\sum x_i^2$ and $\sum x_i$, you can estimate $\mathrm{E}[X^2]$ and $\mathrm{E}[X]$ in a single pass, without having to estimate $\mathrm{E}[X]$ first and then go back for a second pass to calculate $(x_i - \mathrm{E}[X])^2$ for each sample. We won't use this particular application much, but this explains why the formula is popular with statisticians.

5.1.1.2 Variance of a Bernoulli random variable

Recall that a Bernoulli random variable is 1 with probability p and 0 with probability $q = 1 - p$; in particular, any indicator variable is a Bernoulli random variable.

The variance of a Bernoulli random variable is easily calculated from (5.1.2):

$$\begin{aligned}\mathrm{Var}[X] &= \mathrm{E}\left[X^2\right] - (\mathrm{E}[X])^2 \\ &= p - p^2 \\ &= p(1-p) \\ &= pq.\end{aligned}$$

5.1.1.3 Variance of a sum

If $S = \sum_i X_i$, then we can calculate

$$\begin{aligned}\mathrm{Var}[S] &= \mathrm{E}\left[\left(\sum_i X_i\right)^2\right] - \left(\mathrm{E}\left[\sum_i X_i\right]\right)^2 \\ &= \mathrm{E}\left[\sum_i \sum_j X_i X_j\right] - \sum_i \sum_j \mathrm{E}[X_i]\,\mathrm{E}[X_j] \\ &= \sum_i \sum_j \left(\mathrm{E}[X_i X_j] - \mathrm{E}[X_i]\,\mathrm{E}[X_j]\right).\end{aligned}$$

For any two random variables X and Y, the quantity $\mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y]$ is called the **covariance** of X and Y, written $\mathrm{Cov}[X, Y]$. If we take the covariance of a variable and itself, covariance becomes variance: $\mathrm{Cov}[X, X] = \mathrm{Var}[X]$.

We can use $\mathrm{Cov}[X, Y]$ to rewrite the above expansion as

$$\mathrm{Var}\left[\sum_i X_i\right] = \sum_{i,j} \mathrm{Cov}[X_i, X_j] \tag{5.1.3}$$

$$= \sum_i \mathrm{Var}[X_i] + \sum_{i \neq j} \mathrm{Cov}[X_i, X_j] \tag{5.1.4}$$

$$= \sum_i \mathrm{Var}[X_i] + 2\sum_{i<j} \mathrm{Cov}[X_i, X_j] \tag{5.1.5}$$

Note that $\mathrm{Cov}[X, Y] = 0$ when X and Y are independent; this makes Chebyshev's inequality particularly useful for **pairwise-independent** random variables, because then we can just sum up the variances of the individual variables.

A typical application is when we have a sum $S = \sum X_i$ of non-negative random variables with small covariance; here applying Chebyshev's inequality to S can often be used to show that S is not likely to be much smaller than $\mathrm{E}[S]$, which can be handy if we want to show that some lower bound holds on S with some probability. This complements Markov's inequality, which can only be used to get upper bounds.

For example, suppose $S = \sum_{i=1}^{n} X_i$, where the X_i are independent Bernoulli random variables with $\mathrm{E}[X_i] = p$ for all i. Then $\mathrm{E}[S] = np$, and $\mathrm{Var}[S] = \sum_i \mathrm{Var}[X_i] = npq$ (because the X_i are independent). Chebyshev's inequality then says

$$\Pr[|S - \mathrm{E}[S]| \geq \alpha] \leq \frac{npq}{\alpha^2}.$$

The highest variance is when $p = 1/2$. In this case, he probability that S is more than $\beta\sqrt{n}$ away from its expected value $n/2$ is bounded by $\frac{1}{4\beta^2}$. We'll see better bounds on this problem later, but this may already be good enough for many purposes.

More generally, the approach of bounding S from below by estimating $\mathrm{E}[S]$ and either $\mathrm{E}[S^2]$ or $\mathrm{Var}[S]$ is known as the **second-moment method**. In some cases, tighter bounds can be obtained by more careful analysis.

5.1.1.4 Variance of a geometric random variable

Let X be a geometric random variable with parameter p as defined in §3.6.2, so that X takes on the values $1, 2, \ldots$ and $\Pr[X = n] = q^{n-1}p$, where $q = 1 - p$ as usual. What is $\operatorname{Var}[X]$?

We know that $\operatorname{E}[X] = 1/p$, so $(\operatorname{E}[X])^2 = 1/p^2$. Computing $\operatorname{E}[X^2]$ is trickier. Rather than do this directly from the definition of expectation, we can exploit the memorylessness of geometric random variables to get it using conditional expectations, just like we did for $\operatorname{E}[X]$ in §3.6.2.

Conditioning on the two possible outcomes of the first trial, we have

$$\operatorname{E}\left[X^2\right] = p + q \operatorname{E}\left[X^2 \;\middle|\; X > 1\right]. \tag{5.1.6}$$

We now argue that $\operatorname{E}[X^2 \mid X > 1] = \operatorname{E}[(X+1)^2]$. The intuition is that once we have flipped one coin the wrong way, we are back where we started, except that now we have to add that extra coin to X. More formally, we have, for $n > 1$, $\Pr[X^2 = n \mid X > 1] = \frac{\Pr[X^2=n]}{\Pr[X>1]} = \frac{q^{n-1}p}{q} = q^{n-2}p = \Pr[X = n-1] = \Pr[X+1 = n]$. So we get the same probability mass function for X conditioned on $X > 1$ as for $X + 1$ with no conditioning.

Applying this observation to the right-hand side of (5.1.6) gives

$$\begin{aligned}
\operatorname{E}\left[X^2\right] &= p + q \operatorname{E}\left[(X+1)^2\right] \\
&= p + q\left(\operatorname{E}\left[X^2\right] + 2\operatorname{E}[X] + 1\right) \\
&= p + q \operatorname{E}\left[X^2\right] + \frac{2q}{p} + q \\
&= 1 + q \operatorname{E}\left[X^2\right] + \frac{2q}{p}.
\end{aligned}$$

A bit of algebra turns this into

$$\begin{aligned}
\operatorname{E}\left[X^2\right] &= \frac{1 + 2q/p}{1 - q} \\
&= \frac{1 + 2q/p}{p} \\
&= \frac{p + 2q}{p^2} \\
&= \frac{2 - p}{p^2}.
\end{aligned}$$

Now subtract $(\mathrm{E}[X])^2 = \frac{1}{p^2}$ to get

$$\mathrm{Var}[X] = \frac{1-p}{p^2} = \frac{q}{p^2}. \tag{5.1.7}$$

By itself, this doesn't give very good bounds on X. For example, if we want to bound the probability that $X = 1$, we get

$$\begin{aligned}
\Pr[X = 1] &= \Pr\left[X - \mathrm{E}[X] = 1 - \frac{1}{p}\right] \\
&\le \Pr\left[|X - \mathrm{E}[X]| \ge \frac{1}{p} - 1\right] \\
&\le \frac{\mathrm{Var}[X]}{\left(\frac{1}{p} - 1\right)^2} \\
&= \frac{q/p^2}{\left(\frac{1}{p} - 1\right)^2} \\
&= \frac{q}{(1-p)^2} \\
&= \frac{1}{q}.
\end{aligned}$$

Since $\frac{1}{q} \ge 1$, we could have gotten this bound with much less work.

The other direction is not much better. We can easily calculate that $\Pr[X \ge n]$ is exactly q^{n-1} (because this corresponds to flipping n coins the wrong way, no matter what happens with subsequent coins). Using Chebyshev's inequality gives

$$\begin{aligned}
\Pr[X \ge n] &\le \Pr\left[\left|X - \frac{1}{p}\right| \ge n - \frac{1}{p}\right] \\
&\le \frac{q/p^2}{\left(n - \frac{1}{p}\right)^2} \\
&= \frac{q}{(np-1)^2}.
\end{aligned}$$

This at least has the advantage of dropping below 1 when n gets large enough, but it's only polynomial in n while the true value is exponential.

Where this might be useful is in analyzing the sum of a bunch of geometric random variables, as occurs in the Coupon Collector problem discussed in §3.6.3.[1] Letting X_i be the number of balls to take us from $i-1$ to i empty

[1] We are following here a similar analysis in [MU17, §3.3.1].

bins, we have previously argued that X_i has a geometric distribution with $p = \frac{n-i-1}{n}$, so

$$\begin{aligned}\mathrm{Var}\left[X_i\right] &= \frac{i-1}{n} \Bigg/ \left(\frac{n-i-1}{n}\right)^2 \\ &= n\frac{i-1}{(n-i-1)^2},\end{aligned}$$

and

$$\begin{aligned}\mathrm{Var}\left[\sum_{i=1}^{n} X_i\right] &= \sum_{i=1}^{n} \mathrm{Var}\left[X_i\right] \\ &= \sum_{i=1}^{n} n\frac{i-1}{(n-i-1)^2}.\end{aligned}$$

Having the numerator go up while the denominator goes down makes this a rather unpleasant sum to try to solve directly. So we will follow the lead of Mitzenmacher and Upfal and bound the numerator by n, giving

$$\begin{aligned}\mathrm{Var}\left[\sum_{i=1}^{n} X_i\right] &\leq \sum_{i=1}^{n} n\frac{n}{(n-i-1)^2} \\ &= n^2 \sum_{i=1}^{n} \frac{1}{i^2} \\ &\leq n^2 \sum_{i=1}^{\infty} \frac{1}{i^2} \\ &= n^2 \frac{\pi^2}{6}.\end{aligned}$$

The fact that $\sum_{i=1}^{\infty} \frac{1}{i^2}$ converges to $\frac{\pi^2}{6}$ is not trivial to prove, and was first shown by Leonhard Euler in 1735 some ninety years after the question was first proposed.[2] But it's easy to show that the series converges to something, so even if we didn't have Euler's help, we'd know that the variance is $O(n^2)$.

Since the expected value of the sum is $\Theta(n \log n)$, this tells us that we are likely to see a total waiting time reasonably close to this; with at least

[2] See http://en.wikipedia.org/wiki/Basel_Problem for a history, or Euler's original paper [Eul68], available at http://eulerarchive.maa.org/docs/originals/E352.pdf, for the actual proof in the full glory of its original 18th-century typesetting. Curiously, though Euler announced his result in 1735, he didn't submit the journal version until 1749, and it didn't see print until 1768. Things moved more slowly in those days.

a constant probability, it will be within $\Theta(n)$ of the expectation. In fact, the distribution is much more sharply concentrated (see [MU17, §5.4.1] or [MR95, §3.6.3]), but this bound at least gives us something.

5.1.2 More examples

Here are some more examples of Chebyshev's inequality in action. Some of these repeat examples for which we previously got crummier bounds in §4.1.1.

5.1.2.1 Flipping coins

Let X be the sum of n independent fair coins. Let X_i be the indicator variable for the event that the i-th coin comes up heads. Then $\operatorname{Var}[X_i] = 1/4$ and $\operatorname{Var}[X] = \sum \operatorname{Var}[X_i] = n/4$. Chebyshev's inequality gives $\Pr[X = n] \leq \Pr[|X - n/2| \geq n/2] \leq \frac{n/4}{(n/2)^2} = \frac{1}{n}$. This is still not very good, but it's getting better. It's also about the best we can do given only the assumption of pairwise independence.

To see this, let $n = 2^m - 1$ for some m, and let $Y_1 \dots Y_m$ be independent, fair 0–1 random variables. For each non-empty subset S of $\{1 \dots m\}$, let X_S be the exclusive OR of all Y_i for $i \in S$. Then (a) the X_S are pairwise independent; (b) each X_S has variance $1/4$; and thus (c) the same Chebyshev's inequality analysis for independent coin flips above applies to $X = \sum_S X_S$, giving $\Pr[|X - n/2| = n/2] \leq \frac{\operatorname{Var}[S]}{(n/2)^2} = \frac{n/4}{n^2/4} = \frac{1}{n}$. In this case it is not actually possible for X to equal n, but we can have $X = 0$ if all the Y_i are 0, which occurs with probability $2^{-m} = \frac{1}{n+1}$. So the Chebyshev's inequality is almost tight in this case.

5.1.2.2 Balls in bins

Let X_i be the indicator that the i-th of m balls lands in a particular bin. Then $\operatorname{E}[X_i] = 1/n$, giving $\operatorname{E}[\sum X_i] = m/n$, and $\operatorname{Var}[X_i] = 1/n - 1/n^2$, giving $\operatorname{Var}[\sum X_i] = m/n - m/n^2$. So the probability that we get $k + m/n$ or more balls in a particular bin is at most $(m/n - m/n^2)/k^2 < m/nk^2$, and applying the union bound, the probability that we get $k + m/n$ or more balls in any of the n bins is less than m/k^2. Setting this equal to ϵ and solving for k gives a probability of at most ϵ of getting more than $m/n + \sqrt{m/\epsilon}$ balls in any of the bins. This is not as good a bound as we will be able to prove later, but it's at least non-trivial.

5.1.2.3 Lazy select

This example comes from [MR95, §3.3]; essentially the same example, specialized to finding the median, also appears in [MU17, §3.5].[3]

We want to find the k-th smallest element $S_{(k)}$ of a set S of size n. (The parentheses around the index indicate that we are considering the sorted version of the set $S_{(1)} < S_{(2)} \cdots < S_{(n)}$.) The idea is to:

1. Sample a multiset R of $n^{3/4}$ elements of S with replacement and sort them. This takes $O(n^{3/4} \log n^{3/4}) = o(n)$ comparisons so far.

2. Use our sample to find an interval that is likely to contain $S_{(k)}$. The idea is to pick indices $\ell = (k - n^{3/4})n^{-1/4}$ and $r = (k + n^{3/4})n^{-1/4}$ and use $R_{(\ell)}$ and $R_{(r)}$ as endpoints (we are omitting some floors and maxes here to simplify the notation; for a more rigorous presentation see [MR95]). The hope is that the interval $P = [R_{(\ell)}, R_{(r)}]$ in S will both contain $S_{(k)}$, and be small, with $|P| \leq 4n^{3/4} + 2$. We can compute the elements of P in $2n$ comparisons exactly by comparing every element with both $R_{(\ell)}$ and $R_{(r)}$.

3. If both these conditions hold, sort P ($o(n)$ comparisons) and return $S_{(k)}$. If not, try again.

We want to get a bound on how likely it is that P either misses $S_{(k)}$ or is too big.

For any fixed k, the probability that one sample in R is less than or equal to $S_{(k)}$ is exactly k/n, so the expected number X of samples $\leq S_{(k)}$ is exactly $kn^{-1/4}$. The variance on X can be computed by summing the variances of the indicator variables that each sample is $\leq S_{(k)}$, which gives a bound $\operatorname{Var}[X] = n^{3/4}((k/n)(1 - k/n)) \leq n^{3/4}/4$. Applying Chebyshev's inequality gives $\Pr\left[\left|X - kn^{-1/4}\right| \geq \sqrt{n}\right] \leq n^{3/4}/4n = n^{-1/4}/4$.

Now let's look at the probability that P misses $S_{(k)}$ because $R_{(\ell)}$ is too big, where $\ell = kn^{-1/4} - \sqrt{n}$. This is

$$\begin{aligned}\Pr\left[R_{(\ell)} > S_{(k)}\right] &= \Pr\left[X < kn^{-1/4} - \sqrt{n}\right] \\ &\leq n^{-1/4}/4.\end{aligned}$$

[3]The history is that Motwani and Raghavan adapted this algorithm from a similar algorithm by Floyd and Rivest [FR75]. Mitzenmacher and Upfal give a version that also includes the adaptations appearing Motwani and Raghavan, although they don't say where they got it from, and it may be that both textbook versions come from a common folklore source.

(with the caveat that we are being sloppy about round-off errors).

Similarly,

$$\begin{aligned}\Pr\left[R_{(h)} < S_{(k)}\right] &= \Pr\left[X > kn^{-1/4} + \sqrt{n}\right] \\ &\leq n^{-1/4}/4.\end{aligned}$$

So the total probability that P misses $S_{(k)}$ is at most $n^{-1/4}/2$.

Now we want to show that $|P|$ is small. We will do so by showing that it is likely that $R_{(\ell)} \geq S_{(k-2n^{3/4})}$ and $R_{(h)} \leq S_{(k+2n^{3/4})}$. Let X_ℓ be the number of samples in R that are $\leq S_{(k-2n^{3/4})}$ and X_r be the number of samples in R that are $\leq S_{(k+2n^{3/4})}$. Then we have $\mathrm{E}[X_\ell] = kn^{-1/4} - 2\sqrt{n}$ and $\mathrm{E}[X_r] = kn^{-1/4} + 2\sqrt{n}$, and $\mathrm{Var}[X_l]$ and $\mathrm{Var}[X_r]$ are both bounded by $n^{3/4}/4$.

We can now compute

$$\Pr\left[R_{(l)} < S_{(k-2n^{3/4})}\right] = Pr[X_l > kn^{-1/4} - \sqrt{n}] < n^{-1/4}/4$$

by the same Chebyshev's inequality argument as before, and get the symmetric bound on the other side for $\Pr\left[R_{(r)} > S_{(k+2n^{3/4})}\right]$. This gives a total bound of $n^{-1/4}/2$ that P is too big, for a bound of $n^{-1/4} = o(n)$ that the algorithm fails on its first attempt.

The total expected number of comparisons is thus given by $T(n) = 2n + o(n) + O(n^{-1/4}T(n)) = 2n + o(n)$.

5.2 Chernoff bounds

To get really tight bounds, we apply Markov's inequality to $\exp(\alpha S)$, where $S = \sum_i X_i$. This works best when the X_i are independent: if this is the case, so are the variables $\exp(\alpha X_i)$, and so we can easily calculate $\mathrm{E}[\exp(\alpha S)] = \mathrm{E}[\prod_i \exp(\alpha X_i)] = \prod_i \mathrm{E}[\exp(\alpha X_i)]$.

The quantity $\mathrm{E}[\exp(\alpha S)]$, treated as a function of α, is called the **moment generating function** of S, because it expands formally into $\sum_{k=0}^{\infty} \mathrm{E}\left[X^k\right] \frac{\alpha^k}{k!}$, the **exponential generating function** for the series of k-th **moments** $\mathrm{E}\left[X^k\right]$. Note that it may not converge for all S and α;[4] we will be careful to choose α for which it does converge and for which Markov's inequality gives us good bounds.

[4]For example, the moment generating function for our earlier bad X with $\Pr\left[X = 2^k\right] = 2^{-k}$ is equal to $\sum_k 2^{-k} e^{\alpha k}$, which diverges unless $e^\alpha/2 < 1$.

5.2.1 The classic Chernoff bound

The basic Chernoff bound applies to sums of independent 0–1 random variables, which need not be identically distributed. For identically distributed random variables, the sum has a binomial distribution, which we can either compute exactly or bound more tightly using approximations specific to binomial tails; for sums of bounded random variables that aren't necessarily 0–1, we can use Hoeffding's inequality instead (see §5.3).

Let each X_i for $i = 1 \dots n$ be a 0–1 random variable with expectation p_i, so that $\mathrm{E}[S] = \mu = \sum_i p_i$. The plan is to show $\Pr[S \geq (1+\delta)\mu]$ is small when δ and μ are large, by applying Markov's inequality to $\mathrm{E}\left[e^{\alpha S}\right]$, where α will be chosen to make the bound as tight as possible for some specific δ. The first step is to get an upper bound on $\mathrm{E}\left[e^{\alpha S}\right]$.

Compute

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha S}\right] &= \mathrm{E}\left[e^{\alpha \sum X_i}\right] \\
&= \prod_i \mathrm{E}\left[e^{\alpha X_i}\right] \\
&= \prod_i \left(p_i e^{\alpha} + (1-p_i)e^{0}\right) \\
&= \prod_i \left(p_i e^{\alpha} + 1 - p_i\right) \\
&= \prod_i \left(1 + (e^{\alpha} - 1)\, p_i\right) \\
&\leq \prod_i e^{(e^{\alpha}-1)p_i} \\
&= e^{(e^{\alpha}-1)\sum_i p_i} \\
&= e^{(e^{\alpha}-1)\mu}.
\end{aligned}$$

The sneaky inequality step in the middle uses the fact that $(1+x) \leq e^x$ for all x, which itself is one of the most useful inequalities you can memorize.[5]

What's nice about this derivation is that at the end, the p_i have vanished. We don't care what random variables we started with or how many of them there were, but only about their expected sum μ.

Now that we have an upper bound on $\mathrm{E}\left[e^{\alpha S}\right]$, we can throw it into

[5] For a proof of this inequality, observe that the function $f(x) = e^x - (1+x)$ has the derivative $e^x - 1$, which is positive for $x > 0$ and negative for $x < 0$. It follows that $x = 1$ is the unique minimum of f, at which $f(1) = 0$.

Markov's inequality to get the bound we really want:

$$\begin{aligned}\Pr\left[S \geq (1+\delta)\mu\right] &= \Pr\left[e^{\alpha S} \geq e^{\alpha(1+\delta)\mu}\right] \\ &\leq \frac{\mathrm{E}\left[e^{\alpha S}\right]}{e^{\alpha(1+\delta)\mu}} \\ &\leq \frac{e^{(e^{\alpha}-1)\mu}}{e^{\alpha(1+\delta)\mu}} \\ &= \left(\frac{e^{e^{\alpha}-1}}{e^{\alpha(1+\delta)}}\right)^{\mu} \\ &= \left(e^{e^{\alpha}-1-\alpha(1+\delta)}\right)^{\mu}.\end{aligned}$$

We now choose α to minimize the base in the last expression, by minimizing its exponent $e^{\alpha} - 1 - \alpha(1+\delta)$. Setting the derivative of this expression with respect to α to zero gives $e^{\alpha} = (1+\delta)$ or $\alpha = \ln(1+\delta)$; luckily, this value of α is indeed greater than 0 as we have been assuming. Plugging this value in gives

$$\begin{aligned}\Pr\left[S \geq (1+\delta)\mu\right] &\leq \left(e^{(1+\delta)-1-(1+\delta)\ln(1+\delta)}\right)^{\mu} \\ &= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}. \qquad (5.2.1)\end{aligned}$$

The base of this rather atrocious quantity is $e^0/1^1 = 1$ at $\delta = 0$, and its derivative is negative for $\delta \geq 0$ (the easiest way to show this is to substitute $\delta = x - 1$ first). So the bound is never greater than 1 and is both decreasing and less than 1 as soon as $\delta > 0$. We also have that the bound is exponential in μ for any fixed δ.

If we look at the shape of the base as a function of δ, we can observe that when δ is very large, we can replace $(1+\delta)^{1+\delta}$ with δ^{δ} without changing the bound much (and to the extent that we change it, it's an increase, so it still works as a bound). This turns the base into $\frac{e^{\delta}}{\delta^{\delta}} = (e/\delta)^{\delta} = 1/(\delta/e)^{\delta}$. This is pretty close to Stirling's formula for $1/\delta!$ (there is a $\sqrt{2\pi\delta}$ factor missing).

For very small δ, we have that $1 + \delta \approx e^{\delta}$, so the base becomes approximately $\frac{e^{\delta}}{e^{\delta(1+\delta)}} = e^{-\delta^2}$. This approximation goes in the wrong direction (it's smaller than the actual value) but with some fudging we can show bounds of the form $e^{-\mu\delta^2/c}$ for various constants c, as long as δ is not too big.

5.2.2 Easier variants

The full Chernoff bound can be difficult to work with, especially since it's hard to invert (5.2.1) to find a good δ that gives a particular ϵ bound. Fortunately, there are approximate variants that substitute a weaker but less intimidating bound. Some of the more useful are:

- For $0 \leq \delta \leq 1.81$,

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}. \tag{5.2.2}$$

  (The actual upper limit is slightly higher.) Useful for small values of δ, especially because the bound can be inverted: if we want $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\mu\delta^2/3) \leq \epsilon$, we can use any δ with $\sqrt{3\ln(1/\epsilon)/\mu} \leq \delta \leq 1.81$.

  The proof of the approximate bound is to show that, in the given range, $e^\delta/(1+\delta)^{1+\delta} \leq \exp(-\delta^2/3)$. This is easiest to do numerically; a somewhat more formal argument that the bound holds in the range $0 \leq \delta \leq 1$ can be found in [MU17, Theorem 4.4].

- For $0 \leq \delta \leq 4.11$,

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/4}. \tag{5.2.3}$$

  This is a slightly weaker bound than the previous that holds over a larger range. It gives $\Pr[X \geq (1+\delta)\mu] \leq \epsilon$ if $\sqrt{4\ln(1/\epsilon)/\mu} \leq \delta \leq 4.11$. Note that the version given on page 72 of [MR95] is *not correct*; it claims that the bound holds up to $\delta = 2e - 1 \approx 4.44$, but it fails somewhat short of this value.

- For $R \geq 2e\mu$,

$$\Pr[X \geq R] \leq 2^{-R}. \tag{5.2.4}$$

  Sometimes the assumption is replaced with the stronger $R \geq 6\mu$ (this is the version given in [MU17, Theorem 4.4], for example); one can also verify numerically that $R \geq 5\mu$ (i.e., $\delta \geq 4$) is enough. The proof of the $2e\mu$ bound is that $e^\delta/(1+\delta)^{(1+\delta)} < e^{1+\delta}/(1+\delta)^{(1+\delta)} = (e/(1+\delta))^{1+\delta} \leq 2^{-(1+\delta)}$ when $e/(1+\delta) \leq 1/2$ or $\delta \geq 2e-1$. Raising this to μ gives $\Pr[X \geq (1+\delta)\mu] \leq 2^{-(1+\delta)\mu}$ for $\delta \geq 2e-1$. Now substitute R for $(1+\delta)\mu$ (giving $R \geq 2e\mu$) to get the full result. Inverting this one gives $\Pr[X \geq R] \leq \epsilon$ when $R \geq \min(2e\mu, \lg(1/\epsilon))$.

Figure 5.1 shows the relation between the various bounds, in the region where they cross each other.

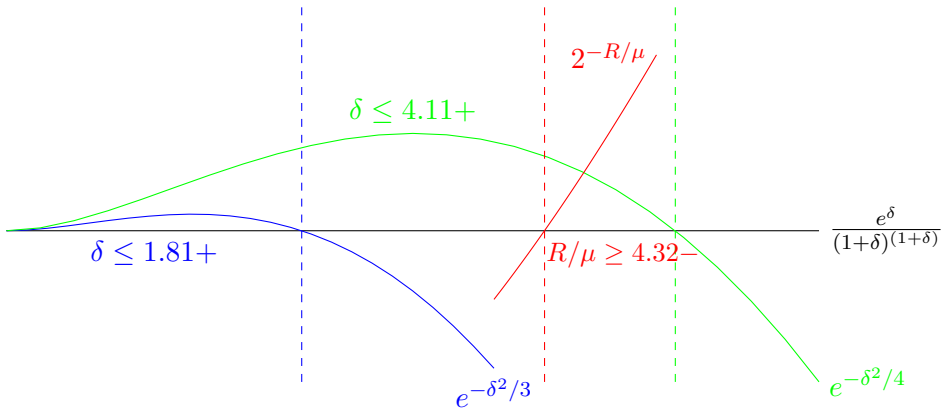


Figure 5.1: Comparison of Chernoff bound variants with exponent μ omitted, plotted in logarithmic scale relative to the standard bound. Each bound is valid only in in the region where it exceeds $e^\delta/(1+\delta)^{1+\delta}$.

5.2.3 Lower bound version

We can also use Chernoff bounds to show that a sum of independent 0–1 random variables isn't too small. The essential idea is to repeat the upper bound argument with a negative value of α, which makes $e^{\alpha(1-\delta)\mu}$ an increasing function in δ. The resulting bound is:

$$\Pr[S \le (1-\delta)\mu] \le \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu. \tag{5.2.5}$$

A simpler but weaker version of this bound is

$$\Pr[S \le (1-\delta)\mu] \le e^{-\mu\delta^2/2}. \tag{5.2.6}$$

Both bounds hold for all δ with $0 \le \delta \le 1$.

5.2.4 Two-sided version

If we combine (5.2.2) with (5.2.6), we get

$$\Pr[|S-\mu| \ge \delta\mu] \le 2e^{-\mu\delta^2/3}, \tag{5.2.7}$$

for $0 \le \delta \le 1.81$.

Suppose that we want this bound to be less than ϵ. Then we need $2e^{-\delta^2/3} \le \epsilon$ or $\delta \ge \sqrt{\frac{3\ln(2/\epsilon)}{\mu}}$. Setting δ to exactly this quantity, (5.2.7)

becomes

$$\Pr\left[|S-\mu| \geq \sqrt{3\mu \ln(2/\epsilon)}\right] \leq \epsilon, \tag{5.2.8}$$

provided $\epsilon \geq 2e^{-\mu/3}$.

For asymptotic purposes, we can omit the constants, giving

Lemma 5.2.1. *Let S be a sum of independent 0–1 variables with $\mathrm{E}[S] = \mu$. Then for any $0 < \epsilon \leq 2e^{-\mu/3}$, S lies within $O\left(\sqrt{\mu \log(1/\epsilon)}\right)$ of μ, with probability at least $1-\epsilon$.*

5.2.5 What if we only have a bound on $\mathrm{E}[S]$?

For some applications, we may not know $\mathrm{E}[S] = \sum \mathrm{E}[X_i]$ exactly, but have only an upper bound $\mathrm{E}[S] \leq \mu$. It is not immediately obvious that we can use this upper bound in place of the actual value of $\mathrm{E}[S]$ when computing an upper tail bound on S, because μ appears in the exponent of all of our bounds, and it is not clear that the corresponding reduction in δ fully compensates for this. However, there is a simple argument that shows that all of these bounds hold even if we overestimate $\mathrm{E}[S]$.

Consider a sum S of independent 0-1 random variables with $\mathrm{E}[S] = \mu' \leq \mu$. We can turn this into a sum of independent 0-1 random variables with expectation exactly μ by adding enough new extra variables to make a second sum T with $\mathrm{E}[T] = \mu - \mu'$. Now $S+T$ satisfies the conditions for (5.2.1), and $S \leq S+T$ always, so

$$\begin{aligned}\Pr[S \geq (1+\delta)\mu] &\leq \Pr[S+T \geq (1+\delta)\mu] \\ &<= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}.\end{aligned}$$

This also works for any of the bounds in §5.2.2 that are derived from (5.2.1).

In the other direction, if we know $\mathrm{E}[S] \geq \mu$ and want to apply the lower tail bound (5.2.5), we can apply a slightly different construction. Suppose that $\mathrm{E}[S] = \mu' \geq \mu$. For each X_i, construct a 0-1 random variable Y_i such that (a) all the Y_i are independent of each other, (b) $Y_i \leq X_i$ always, and (c) $\mathrm{E}[Y_i] = \mathrm{E}[X_i](\mu/\mu')$. The easiest way to do this is to set $Y_i = X_i Z_i$ where each Z_i is an independent biased coin with expectation μ/μ'.

Let $T = \sum Y_i$. Then $T \leq S$ and $\mathrm{E}[T] = \sum \mathrm{E}[Y_i] = \sum \mathrm{E}[X_i](\mu/\mu') = \mu$.

Since T satisfies the requirements of (5.2.5), we can argue

$$\begin{aligned}\Pr[S \le (1-\delta)\mu] &\le \Pr[T \le (1-\delta)\mu] \\ &\le \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}.\end{aligned}$$

As with the upper tail bound, this approach also works for simpified versions of the lower tail bound like (5.2.6).

For the two-sided variants, we are out of luck. The best we can do if we know $a \le \mathrm{E}[S] \le b$ is to apply each of the one-sided bounds separately.

5.2.6 Almost-independent variables

Chernoff bounds generally don't work very well for variables that are not independent, and in most such cases we must use Chebyshev's inequality (§5.1) or the Azuma-Hoeffding inequality (§5.3) instead. But there is one special case that comes up occasionally where it helps to be able to apply the Chernoff bound to variables that are *almost* independent in a particular technical sense.

Lemma 5.2.2. *Let $X_1, \dots, X_n$ be 0–1 random variables with the property that $\mathrm{E}[X_i \mid X_1, \dots, X_{i-1}] \le p_i \le 1$ for all i. Let $\mu = \sum_{i=1}^n p_i$ and $S = \sum_{i=1}^n X_i$. Then (5.2.1) holds.*

Alternatively, let $\mathrm{E}[X_i \mid X_1, \dots, X_{i-1}] \ge p_i \ge 0$ for all i, and let $\mu = \sum_{i=1}^n p_i$ and $S = \sum_{i=1}^n X_i$ as before. Then (5.2.5) holds.

Proof. Rather than repeat the argument for independent variables, we will employ a coupling, where we replace the X_i with independent Y_i so that $\sum_{i=1}^n Y_i$ gives a bound an $\sum_{i=1}^n X_i$.

For the upper bound, let each $Y_i = 1$ with independent probability p_i. Use the following process to generate a new X_i' in increasing order of i: if $Y_i = 0$, set $X_i' = 0$. Otherwise set $X_i' = 1$ with probability $\Pr[X_i = 1 \mid X_1 = X_1', \dots X_{i-1}'] / p_i$, Then $X_i' \le Y_i$, and

$$\begin{aligned}\Pr[X_i' = 1 \mid X_1', \dots, X_i'] &= \left(\Pr[X_i = 1 \mid X_1 = X_1', \dots, X_{i-1} = X_{i-1}'] / p_i\right) \Pr[Y_i = 1] \\ &= \Pr[X_i = 1 \mid X_1 = X_1', \dots, X_{i-1} = X_{i-1}'].\end{aligned}$$

It follows that the X_i' have the same joint distribution as the X_i, and so

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} X_i' \geq \mu(1+\delta)\right] &= \Pr\left[\sum_{i=1}^{n} X_i \geq \mu(1+\delta)\right] \\ &\leq \Pr\left[\sum_{i=1}^{n} Y_i \geq \mu(1+\delta)\right] \\ &\leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu.\end{aligned}$$

For the other direction, generate the X_i first and generate the Y_i using the same rejection sampling trick. Now the Y_i are independent (because their joint distribution is) and each Y_i is a lower bound on the corresponding X_i. □

The lemma is stated for the general Chernoff bounds (5.2.1) and (5.2.5), but the easier versions follow from these, so they hold as well, as long as we are careful to remember that the μ in the upper bound is not necessarily the same μ as in the lower bound.

5.2.7 Other tail bounds for the binomial distribution

The random graph literature can be a good source for bounds on the binomial distribution. See for example [Bol01, §1.3], which uses normal approximation to get bounds that are slightly tighter than Chernoff bounds in some cases, and [JLR00, Chapter 2], which describes several variants of Chernoff bounds as well as tools for dealing with sums of random variables that aren't fully independent.

5.2.8 Applications

5.2.8.1 Flipping coins

Suppose S is the sum of n independent fair coin-flips. Then $\mathrm{E}[S] = n/2$ and $\Pr[S = n] = \Pr[S \geq 2\,\mathrm{E}[S]]$ is bounded using (5.2.1) by setting $\mu = n/2$, $\delta = 1$ to get $\Pr[S = n] \leq (e/4)^{n/2} = (2/\sqrt{e})^{-n}$. This is not quite as good as the real answer 2^{-n} (the quantity $2/\sqrt{e}$ is about 1.213...), but it's at least exponentially small.

5.2.8.2 Balls in bins again

Let's try applying the Chernoff bound to the balls-in-bins problem. Here we let $S = \sum_{i=1}^{m} X_i$ be the number of balls in a particular bin, with X_i the indicator that the i-th ball lands in the bin, $\mathrm{E}[X_i] = p_i = 1/n$, and $\mathrm{E}[S] = \mu = m/n$. To get a bound on $\Pr[S \geq m/n + k]$, apply the Chernoff bound with $\delta = kn/m$ to get

$$\begin{aligned}\Pr[S \geq m/n + k] &= \Pr[S \geq (m/n)(1 + kn/m)] \\ &\leq \frac{e^k}{(1 + kn/m)^{1+kn/m}}.\end{aligned}$$

For $m = n$, this collapses to the somewhat nicer but still pretty horrifying $e^k/(k+1)^{k+1}$.

Staying with $m = n$, if we are bounding the probability of having large bins, we can use the 2^{-R} variant to show that the probability that any particular bin has more than $2 \lg n$ balls (for example), is at most n^{-2}, giving the probability that there exists such a bin of at most $1/n$. This is not as strong as what we can get out of the full Chernoff bound. If we take the logarithm of $e^k/(k+1)^{k+1}$, we get $k - (k+1)\ln(k+1)$; if we then substitute $k = \frac{c \ln n}{\ln \ln n} - 1$, we get

$$\begin{aligned}&\frac{c \ln n}{\ln \ln n} - 1 - \frac{c \ln n}{\ln \ln n} \ln \frac{c \ln n}{\ln \ln n} \\ &= (\ln n)\left(\frac{c}{\ln \ln n} - \frac{1}{\ln n} - \frac{c}{\ln \ln n}(\ln c + \ln \ln n - \ln \ln \ln n)\right) \\ &= (\ln n)\left(\frac{c}{\ln \ln n} - \frac{1}{\ln n} - \frac{c \ln c}{\ln \ln n} - c + \frac{c \ln \ln \ln n}{\ln \ln n}\right) \\ &= (\ln n)(-c + o(1)).\end{aligned}$$

So the probability of getting more than $c \ln n / \ln \ln n$ balls in any one bin is bounded by $\exp((\ln n)(-c + o(1))) = n^{-c+o(1)}$. This gives a maximum bin size of $O(\log n / \log \log n)$ with any fixed probability bound n^{-a} for sufficiently large n.

5.2.8.3 Flipping coins, central behavior

Suppose we flip n fair coins, and let S be the number that come up heads. We expect $\mu = n/2$ heads on average. How many extra heads can we get, if we want to stay within a probability bound of n^{-c}?

Here we use the small-δ approximation, which gives $\Pr[S \geq (1+\delta)(n/2)] \leq \exp(-\delta^2 n/6)$. Setting $\exp(-\delta^2 n/6) = n^{-c}$ gives $\delta = \sqrt{6 \ln n^c / n} = \sqrt{6c \ln n / n}$.

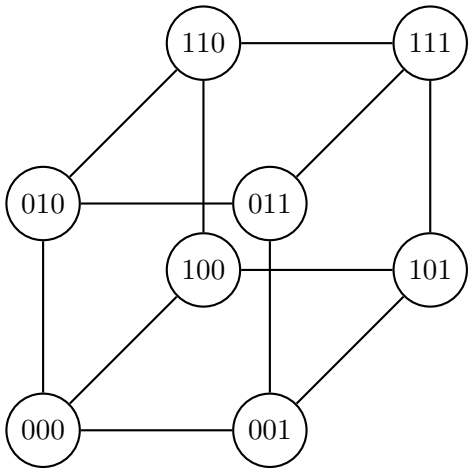


Figure 5.2: Hypercube network with $n = 3$

The actual excess over the mean is $\delta(n/2) = (n/2)\sqrt{6c \ln n/n} = \sqrt{\frac{3}{2}cn \ln n}$. By symmetry, the same bound applies to extra tails. So if we flip 1000 coins and see more than 676 heads (roughly the bound when c=3), we can reasonably conclude that either (a) our coin is biased, or (b) we just hit a rare one-in-a-billion jackpot.

In algorithm analysis, the $\sqrt{(3/2)c}$ part usually gets absorbed into the asymptotic notation, and we just say that with probability at least $1 - n^{-c}$, the sum of n random bits is within $O(\sqrt{n \log n})$ of $n/2$.

5.2.8.4 Permutation routing on a hypercube

Here we use Chernoff bounds to show bounds on a classic **permutation-routing** algorithm for **hypercube networks** due to Valiant [Val82]. The presentation here is based on §4.2 of [MR95], which in turn is based on an improved version of Valiant's original analysis that appeared in a follow-up paper with Brebner [VB81]. There's also a write-up of this in [MU17, §4.6.1].

The basic idea of a hypercube architecture is that we have a collection of $N = 2^n$ processors, each with an n-bit address. Two nodes are adjacent if their addresses differ by one bit (see Figure 5.2 for an example). Though now mostly of theoretical interest, these things were the cat's pajamas back in the 1980s: see http://en.wikipedia.org/wiki/Connection_Machine.

Suppose that at some point in a computation, each processor i wants to send a packet of data to some processor $\pi(i)$, where π is a permutation of the addresses. But we can only send one packet per time unit along each of

the n edges leaving a processor.[6] How do we route the packets so that all of them arrive in the minimum amount of time?

We could try to be smart about this, or we could use randomization. Valiant's idea is to first route each process i's packet to some random intermediate destination $\sigma(i)$, then in the second phase, we route it from $\sigma(i)$ to its ultimate destination $\pi(i)$. Unlike π, σ is not necessarily a permutation; instead, $\sigma(i)$ is chosen uniformly at random independently of all the other $\sigma(j)$. This makes the choice of paths for different packets independent of each other, which we will need later to apply Chernoff bounds..

Routing is done by a **bit-fixing**: if a packet is currently at node x and heading for node y, find the leftmost bit j where $x_j \neq y_j$ and fix it, by sending the packet on to $x[x_j/y_j]$. In the absence of contention, bit-fixing routes a packet to its destination in at most n steps. The hope is that the randomization will tend to spread the packets evenly across the network, reducing the contention for edges enough that the actual time will not be much more than this.

The first step is to argue that, during the first phase, any particular packet is delayed at most one time unit by any other packet whose path overlaps with it. Suppose packet i is delayed by contention on some edge uv. Then there must be some other packet j that crosses uv during this phase. From this point on, j remains one step ahead of i (until its path diverges), so it can't block i again unless both are blocked by some third packet k (in which case we charge i's further delay to k). This means that we can bound the delays for packet i by counting how many other packets cross its path.[7] So now we just need a high-probability bound on the number of packets that get in a particular packet's way.

Following the presentation in [MR95], define H_{ij} to be the indicator variable for the event that packets i and j cross paths during the first phase. Because each j chooses its destination independently, once we fix i's path, the H_{ij} are all independent. So we can bound $S = \sum_{j \neq i} H_{ij}$ using Chernoff bounds. To do so, we must first calculate an upper bound on $\mu = \mathrm{E}[S]$.

The trick here is to observe that any path that crosses i's path must cross one of its edges, and we can bound the number of such paths by bounding how many paths cross each edge. For each edge e, let T_e be the number of paths that cross edge e, and for each j, let X_j be the number of edges that path j crosses. Counting two ways, we have $\sum_e T_e = \sum_j X_j$, and so

[6] Formally, we have a synchronous routing model with unbounded buffers at each node, with a maximum capacity of one packet per edge per round.

[7] A much more formal version of this argument is given as [MR95, Lemma 4.5].

$\mathrm{E}\left[\sum_e T_e\right] = \mathrm{E}\left[\sum_j X_j\right] \leq N(n/2)$. By symmetry, all the T_e have the same expectation, so we get $\mathrm{E}[T_e] \leq \frac{N(n/2)}{Nn} = 1/2$.

Now fix $\sigma(i)$. This determines some path $e_1 e_2 \dots e_k$ for packet i. In general we do not expect $\mathrm{E}[T_{e_\ell} \mid \sigma(i)]$ to equal $\mathrm{E}[T_{e_\ell}]$, because conditioning on i's path crossing e_ℓ guarantees that at least one path crosses this edge that might not have. However, if we let T'_e be the number of packets $j \neq i$ that cross e, then we have $T'_e \leq T_e$ always, giving $\mathrm{E}[T'_e] \leq \mathrm{E}[T_e]$, and because T'_e does not depend on i's path, $\mathrm{E}[T'_e \mid \sigma(i)] = \mathrm{E}[T'_e] \leq \mathrm{E}[T_e] \leq 1/2$. Summing this bound over all $k \leq n$ edges on i's path gives $\mathrm{E}\left[\sum_{j\neq i} H_{ij} \;\middle|\; \sigma(i)\right] \leq n/2$, which implies $\mathrm{E}\left[\sum_{j\neq i} H_{ij}\right] \leq n/2$ after removing the conditioning on $\sigma(i)$.

Inequality (5.2.4) says that $\Pr[X \geq R] \leq 2^{-R}$ when $R \geq 2e\mu$. Letting $X = \sum_{j\neq i} H_{ij}$ and setting $R = 3n$ gives $R = 6(n/2) \geq 6\mu > 2e\mu$, so $\Pr\left[\sum_j H_{ij} \geq 3n\right] \leq 2^{-3n} = N^{-3}$. This says that any one packet reaches its random destination with at most $3n$ added delay (thus, in at most $4n$ time units) with probability at least $1 - N^{-3}$. If we consider all N packets, the total probability that any of them fail to reach their random destinations in $4n$ time units is at most $N \cdot N^{-3} = N^{-2}$. Note that because we are using the union bound, we don't need independence for this step—which is good, because we don't have it.

What about the second phase? Here, routing the packets from the random destinations back to the real destinations is just the reverse of routing them from the real destinations to the random destinations. So the same bound applies, and with probability at most N^{-2} some packet takes more than $4n$ time units to get back (this assumes that we hold all the packets before sending them back out, so there are no collisions between packets from different phases).

Adding up the failure probabilities and costs for both stages gives a probability of at most $2/N^2$ that any packet takes more than $8n$ time units to reach its destination.

The structure of this argument is pretty typical for applications of Chernoff bounds: we get a very small bound on the probability that something bad happens by applying Chernoff bounds to a part of the problem where we have independence, then use the union bound to extend this to the full problem where we don't.

5.3 The Azuma-Hoeffding inequality

The problem with Chernoff bounds is that they only work for sums of Bernoulli random variables. **Hoeffding's inequality** [Hoe63] is another concentration bound based on the moment generating function that applies to any sum of bounded independent random variables with mean 0.[8] It has the additional useful feature that it generalizes nicely to some collections of random variables that are not mutually independent, as we will see in §5.3.2. This more general version is known as **Azuma's inequality** [Azu67] or the **Azuma-Hoeffding inequality**.[9]

5.3.1 Hoeffding's inequality

This is the version for sums of bounded independent random variables. We will consider the symmetric case, where each variable X_i satisfies $|X_i| \leq c_i$ for some constant c_i. Hoeffding's original result considered bounds of the form $a_i \leq X_i \leq b_i$, and is equivalent when $a_i = -b_i$.

The main tool is **Hoeffding's lemma**, which states

Lemma 5.3.1. *Let* $\mathrm{E}[X] = 0$ *and* $|X| \leq c$ *with probability 1. Then*

$$\mathrm{E}\left[e^{\alpha X}\right] \leq e^{(\alpha c)^2/2}. \tag{5.3.1}$$

Proof. The basic idea is that, for any α, $e^{\alpha x}$ is a convex function. Since we want an upper bound, we can't use Jensen's inequality (4.3.1), but we *can* use the fact that X is bounded and we know its expectation. Convexity of $e^{\alpha x}$ means that, for any x with $-c \leq x \leq c$, $e^{\alpha x} \leq \lambda e^{-\alpha c} + (1-\lambda)e^{\alpha c}$, where $x = \lambda(-c) + (1-\lambda)c$. Solving for λ in terms of x gives $\lambda = \frac{1}{2}\left(1 - \frac{x}{c}\right)$ and $1 - \lambda = \frac{1}{2}\left(1 + \frac{x}{c}\right)$. So

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha X}\right] &\leq \mathrm{E}\left[\frac{1}{2}\left(1 - \frac{X}{c}\right)e^{-\alpha c} + \frac{1}{2}\left(1 + \frac{X}{c}\right)e^{\alpha c}\right] \\
&= \frac{e^{-\alpha c} + e^{\alpha c}}{2} - \frac{e^{-\alpha c}}{2c}\mathrm{E}[X] + \frac{e^{\alpha c}}{2c}\mathrm{E}[X] \\
&= \frac{e^{-\alpha c} + e^{\alpha c}}{2} \\
&= \cosh(\alpha c).
\end{aligned}$$

[8] Note that the requirement that $\mathrm{E}[X_i] = 0$ can always be satisfied by considering instead $Y_i = X_i - \mathrm{E}[X_i]$.

[9] The history of this is that Hoeffding [Hoe63] proved it for independent random variables, and observed that the proof was easily extended to martingales, while Azuma [Azu67] actually went and did the work of proving it for martingales.

In other words, the worst possible X is a fair choice between $\pm c$, and in this case we get the hyperbolic cosine of αc as its moment generating function.

We don't like hyperbolic cosines much, because we are going to want to take products of our bounds, and hyperbolic cosines don't multiply very nicely. As before with $1 + x$, we'd be much happier if we could replace the cosh with a nice exponential. The Taylor series expansion of $\cosh x$ starts with $1+x^2/2+\ldots$, suggesting that we should approximate it with $\exp(x^2/2)$, and indeed it is the case that for all x, $\cosh x \leq e^{x^2/2}$. This can be shown by comparing the rest of the Taylor series expansions:

$$\begin{aligned}
\cosh x &= \frac{e^x + e^{-x}}{2} \\
&= \frac{1}{2}\left(\sum_{n=0}^{\infty} \frac{x^n}{n!} + \sum_{n=0}^{\infty} \frac{(-x)^n}{n!}\right) \\
&= \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} \\
&\leq \sum_{n=0}^{\infty} \frac{x^{2n}}{2^n n!} \\
&= \sum_{n=0}^{\infty} \frac{(x^2/2)^n}{n!} \\
&= e^{x^2/2}.
\end{aligned}$$

This gives the claimed bound

$$\mathrm{E}\left[e^{\alpha X}\right] \leq \cosh(\alpha c) \leq e^{(\alpha c)^2/2}.$$

□

.

Theorem 5.3.2. *Let $X_1 \ldots X_n$ be independent random variables with $\mathrm{E}[X_i] = 0$ and $|X_i| \leq c_i$ for all i. Then for all t,*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{5.3.2}$$

Proof. Let $S = \sum_{i=1}^{n} X_i$. As with Chernoff bounds, we'll first calculate a bound on the moment generating function $\mathrm{E}\left[e^{\alpha S}\right]$ and then apply Markov's inequality with a carefully-chosen α.

From (5.3.1), we have $\mathrm{E}\left[e^{\alpha X_i}\right] \leq e^{(\alpha c_i)^2/2}$ for all i. Using this bound and the independence of the X_i, we compute

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha S}\right] &= \mathrm{E}\left[\exp\left(\alpha \sum_{i=1}^{n} X_i\right)\right] \\
&= \mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] \\
&= \prod_{i=1}^{n} \mathrm{E}\left[e^{\alpha X_i}\right] \\
&\leq \prod_{i=1}^{n} e^{(\alpha c_i)^2/2} \\
&= \exp\left(\sum_{i=1}^{n} \frac{\alpha^2 c_i^2}{2}\right) \\
&= \exp\left(\frac{\alpha^2}{2} \sum_{i=1}^{n} c_i^2\right).
\end{aligned}$$

Applying Markov's inequality then gives (when $\alpha > 0$):

$$\begin{aligned}
\Pr[S \geq t] &= \Pr\left[e^{\alpha S} \geq e^{\alpha t}\right] \\
&\leq \exp\left(\frac{\alpha^2}{2} \sum_{i=1}^{n} c_i^2 - \alpha t\right). \qquad (5.3.3)
\end{aligned}$$

Now we do the same trick as in Chernoff bounds and choose α to minimize the bound. If we write C for $\sum_{i=1}^{n} c_i^2$, this is done by minimizing the exponent $\frac{\alpha^2}{2}C - \alpha t$, which we do by taking the derivative with respect to α and setting it to zero: $\alpha C - t = 0$, or $\alpha = t/C$. At this point, the exponent becomes $\frac{(t/C)^2}{2}C - (t/C)t = -\frac{t^2}{2C}$.

Plugging this into (5.3.3) gives the bound (5.3.2) claimed in the theorem. □

5.3.1.1 Hoeffding vs Chernoff

Let's see how good a bound this gets us for our usual test problem of bounding $\Pr[S = n]$ where $S = \sum_{i=1}^{n} X_i$ is the sum of n independent fair coin-flips. To make the problem fit the theorem, we replace each X_i by a rescaled version $Y_i = 2X_i - 1 = \pm 1$ with equal probability; this makes $\mathrm{E}[Y_i] = 0$ as needed,

with $|Y_i| \leq c_i = 1$. Hoeffding's inequality (5.3.2) then gives

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} Y_i \geq n\right] &\leq \exp\left(-\frac{n^2}{2n}\right) \\ &= e^{-n/2} = (\sqrt{e})^{-n}.\end{aligned}$$

Since $\sqrt{e} \approx 1.649\ldots$, this is actually slightly better than the $(2/\sqrt{e})^{-n}$ bound we get using Chernoff bounds.

On the other hand, Chernoff bounds work better if we have a more skewed distribution on the X_i; for example, in the balls-in-bins case, each X_i is a Bernoulli random variable with $\mathrm{E}[X_i] = 1/n$. Using Hoeffding's inequality, we get a bound c_i on $|X_i - \mathrm{E}[X_i]|$ of only $1 - 1/n$, which puts $\sum_{i=1}^{n} c_i^2$ very close to n, requiring $t = \Omega(\sqrt{n})$ before we get any non-trivial bound out of (5.3.2), pretty much the same as in the fair-coin case (which is not surprising, since Hoeffding's inequality doesn't know anything about the distribution of the X_i). But we've already seen that Chernoff gives us that $\sum X_i = O(\log n / \log\log n)$ with high probability in this case.

5.3.1.2 Asymmetric version

The original version of Hoeffding's inequality [Hoe63] assumes $a_i \leq X_i \leq b_i$, but $\mathrm{E}[X_i]$ is still zero for all X_i. In this version, the bound is

$$\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.3.4}$$

This reduces to (5.3.2) when $a_i = -c_i$ and $b_i = c_i$. The proof is essentially the same, but a little more analytic sneakery is required to show that $\mathrm{E}\left[e^{\alpha X_i}\right] \leq e^{\alpha^2 (b_i - a_i)^2/8}$; see [McD89] for a proof of this that is a little more approachable than Hoeffding's original paper. For most applications, the only difference between the symmetric version (5.3.2) and the asymmetric version (5.3.4) is a small constant factor on the resulting bound on t.

Hoeffding's inequality is not the tightest possible inequality that can be obtained from the conditions under which it applies, but is relatively simple and easy to work with. For a particularly strong version of Hoeffding's inequality and a discussion of some variants, see [FGQ12].

5.3.2 Azuma's inequality

A general rule of thumb is that most things that work for sums of independent random variables also work for martingales, which are sequences of random variables that have similar behavior but allow for more dependence.

Formally, a **martingale** is a sequence of random variables $S_0, S_1, S_2, \ldots$, where $\mathrm{E}[S_t \mid S_1, \ldots, S_{t-1}] = S_{t-1}$. In other words, given everything you know up until time $t-1$, your best guess of the expected value at time t is just wherever you are now.

Another way to describe a martingale is to take the partial sums $S_t = \sum_{i=1}^{t} X_t$ of a **martingale difference sequence**, which is a sequence of random variables $X_1, X_2, \ldots$ where $\mathrm{E}[X_t \mid X_1 \ldots X_{t-1}] = 0$. So in this version, your expected change from time $t-1$ to t averages out to zero, even if you try to predict it using all the information you have at time $t-1$.

In some cases it makes sense to allow extra information to sneak in. We can represent this using σ-algebras, in particular by using a filtration of the form $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$, where each $\mathcal{F}_t$ is a σ-algebra (see §3.5.3). A sequence $S_0, S_1, S_2, \ldots$ is **adapted** to a filtration $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$ if each S_t is $\mathcal{F}_t$-measurable. This means that at time t the sum of our knowledge ($\mathcal{F}_t$) is enough to predict exactly the value of S_t. The subset relations also mean that we remember everything there is to know about $S_{t'}$ for $t' < t$.

The general definition of a martingale is a collection $\{(S_t, \mathcal{F}_t) \mid t \in \mathbb{N}\}$ where

1. Each S_t is $\mathcal{F}_t$-measurable; and
2. $\mathrm{E}[S_{t+1} \mid \mathcal{F}_t] = S_t$.

This means that even if we include any extra information we might have at time t, we still can't predict S_{t+1} any better than by guessing the current value S_t. This alternative definition will be important in some special cases, as when S_t is a function of some other collection of random variables that we use to define the $\mathcal{F}_t$. Because $\mathcal{F}_t$ includes at least as much information as $S_0, \ldots, S_t$, it will always be the case that any sequence $\{(S_t, \mathcal{F}_t)\}$ that is a martingale in the general sense gives a sequence $\{S_t\}$ that is a martingale in the more specialized $\mathrm{E}[S_{t+1} \mid S_0, \ldots, S_t] = S_t$ sense.

Martingales were invented to analyze fair gambling games, where your return over some time interval is not independent of previous outcomes (for example, you may change your bet or what game you are playing depending on how things have been going for you), but it is always zero on average given previous information.[10] The nice thing about martingales is they allow

[10] Real casinos give negative expected return, so your winnings in a real casino form a **supermartingale** with $S_t \geq \mathrm{E}[S_{t+1} \mid S_0 \ldots S_t]$. On the other hand, the casino's take, in a well-run casino, is a **submartingale**, a process with $S_t \leq \mathrm{E}[S_{t+1} \mid S_0 \ldots S_t]$. These definitions also generalize in the obvious way to the $\{(S_t, \mathcal{F}_t)\}$ case.

for a bit of dependence while still acting very much like sums of independent random variables.

Where this comes up with Hoeffding's inequality is that we might have a process that is reasonably well-behaved, but its increments are not technically independent. For example, suppose that a gambler plays a game where they bet x units $0 \leq x \leq 1$ at each round, and receives $\pm x$ with equal probability. Suppose also that their bet at each round may depend on the outcome of previous rounds (for example, they might stop betting entirely if they lose too much money). If X_i is their take at round i, we have that $\mathrm{E}[X_i \mid X_1 \dots X_{i-1}] = 0$ and that $|X_i| \leq 1$. This is enough to apply the martingale version of Hoeffding's inequality, often called Azuma's inequality.

Theorem 5.3.3. *Let $\{S_k\}$ be a martingale with $S_k = \sum_{i=1}^{k} X_i$ and $|X_i| \leq c_i$ for all i. Then for all n and all $t \geq 0$:*

$$\Pr[S_n \geq t] \leq \exp\left(\frac{-t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{5.3.5}$$

Proof. Basically, we just show that $\mathrm{E}\left[e^{\alpha S_n}\right] \leq \exp\left(\frac{\alpha^2}{2}\sum_{i=1}^{n} c_i^2\right)$—just like in the proof of Theorem 5.3.2—and the rest follows using the same argument. The only tricky part is we can no longer use independence to transform $\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right]$ into $\prod_{i=1}^{n} \mathrm{E}\left[e^{\alpha X_i}\right]$.

Instead, we use the martingale property. For each X_i, we have $\mathrm{E}[X_i \mid X_1 \dots X_{i-1}] = 0$ and $|X_i| \leq c_i$ always. Recall that $\mathrm{E}\left[e^{\alpha X_i} \,\middle|\, X_1 \dots X_{i-1}\right]$ is a random variable that takes on the average value of $e^{\alpha X_i}$ for each setting of $X_1 \dots X_{i-1}$. We can apply the same analysis as in the proof of 5.3.2 to show that this means that $\mathrm{E}\left[e^{\alpha X_i} \,\middle|\, X_1 \dots X_{i-1}\right] \leq e^{(\alpha c_i)^2/2}$ always.

The trick is to use the fact that, for any random variables X and Y, $\mathrm{E}[XY] = \mathrm{E}[\mathrm{E}[XY \mid X]] = \mathrm{E}[X\,\mathrm{E}[Y \mid X]]$.

We argue by induction on n that $\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] \leq \prod_{i=1}^{n} e^{(\alpha c)^2/2}$. The

base case is when $n = 0$. For the induction step, compute

$$\begin{aligned}
\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i}\right] &= \mathrm{E}\left[\mathrm{E}\left[\prod_{i=1}^{n} e^{\alpha X_i} \;\middle|\; X_1 \dots X_{n-1}\right]\right] \\
&= \mathrm{E}\left[\left(\prod_{i=1}^{n} e^{\alpha X_i}\right) \mathrm{E}\left[e^{\alpha X_n} \;\middle|\; X_1 \dots X_{n-1}\right]\right] \\
&\leq \mathrm{E}\left[\left(\prod_{i=1}^{n-1} e^{\alpha X_i}\right) e^{(\alpha c_n)^2/2}\right] \\
&= \mathrm{E}\left[\prod_{i=1}^{n-1} e^{\alpha X_i}\right] e^{(\alpha c_n)^2/2} \\
&\leq \left(\prod_{i=1}^{n-1} e^{(\alpha c_i)^2/2}\right) e^{(\alpha c_n)^2/2} \\
&= \prod_{i=1}^{n} e^{(\alpha c_i)^2/2} \\
&= \exp\left(\frac{\alpha^2}{2} \sum_{i=1}^{n} c_i^2\right).
\end{aligned}$$

The rest of the proof goes through as before. □

Some extensions:

- The asymmetric version of Hoeffding's inequality (5.3.4) also holds for martingales. So if each increment X_i satisfies $a_i \leq X_i \leq b_i$ always,

$$\Pr\left[\sum_{i=1}^{n} X_i \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{5.3.6}$$

- The same bound works for bounded-difference **supermartingales**. A supermartingale is a process where $\mathrm{E}[X_i \mid X_1 \dots X_{i-1}] \leq 0$; the idea is that my expected gain at any step is non-positive, so my present wealth is always superior to my future wealth.[11] If $\mathrm{E}[X_i \mid X_1 \dots X_{i-1}] \leq 0$ and $|X_i| \leq c_i$, then we can write $X_i = Y_i + Z_i$ where $Y_i = \mathrm{E}[X_i \mid X_1 \dots X_{i-1}] \leq 0$ is predictable from $X_1 \dots X_{i-1}$ and $\mathrm{E}[Z_i \mid X_1 \dots X_{i-1}] = 0$.[12] Then we can bound $\sum_{i=1}^{n} X_i$ by observing that it is no greater than $\sum_{i=1}^{n} Z_i$.

[11] The corresponding notion in the other direction is a **submartingale**. See §9.2.

[12] This is known as a **Doob decomposition** and can be used to extract a martingale $\{Z_i\}$ from any stochastic process $\{X_i\}$. For general processes, $Y_i = X_i - Z_i$ will still be predictable, but may not satisfy $\mathrm{E}[Y_i \mid X_1, \dots, X_{i-1}] \leq 0$.

A complication is that we no longer have $|Z_i| \le c_i$; instead, $|Z_i| \le 2c_i$ (since leaving out Y_i may shift Z_i up). But with this revised bound, (5.3.5) gives

$$\begin{aligned}\Pr\left[\sum_{i=1}^{n} X_i \ge t\right] &\le \Pr\left[\sum_{i=1}^{n} Z_i \ge t\right] \\ &\le \exp\left(-\frac{t^2}{8\sum_{i=1}^{n} c_i^2}\right). \end{aligned} \tag{5.3.7}$$

- Suppose that we stop the process after the first time τ with $S_\tau = \sum_{i=1}^{\tau} X_i \ge t$. This is equivalent to making a new variable Y_i that is zero whenever $S_{i-1} \ge t$ and equal to X_i otherwise. This doesn't affect the conditions $\mathrm{E}[Y_i \mid Y_1 \dots Y_{i-1}] = 0$ or $|Y_i| \le c_i$, but it makes it so $\sum_{i=1}^{n} Y_i \ge t$ if and only if $\max_{k \le n} \sum_{i=1}^{k} X_i \ge t$. Applying (5.3.5) to $\sum Y_i$ then gives

$$\Pr\left[\max_{k \le n} \sum_{i=1}^{k} X_i \ge t\right] \le \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{5.3.8}$$

- Since the conditions on X_i in Theorem 5.3.3 apply equally well to $-X_i$, we have

$$\Pr\left[\sum_{i=1}^{n} X_i \le -t\right] \le \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{5.3.9}$$

  which we can combine with (5.3.5) to get the two-sided bound

$$\Pr\left[\left|\sum_{i=1}^{n} X_i\right| \ge t\right] \le 2\exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right). \tag{5.3.10}$$

- The extension of Hoeffding's inequality to the case $a_i \le X_i \le b_i$ works equally well for Azuma's inequality, giving the same bound as in (5.3.4).

- Finally, one can replace the requirement that each c_i be a constant with a requirement that c_i be predictable from $X_1 \dots X_{i-1}$ and that $\sum_{i=1}^{n} c_i^2 \le C$ always and get $\Pr[\sum_{i=1}^{n} X_i \ge t] \le e^{-t^2/2C}$. This generally doesn't come up unless you have an algorithm that explicitly cuts off the process if $\sum c_i^2$ gets too big, but there is at least one example of this in the literature [AW96].

There are also cases where the asymmetric version works with $a_i \leq X_i \leq b_i$ where a bound on $b_i - a_i$ is fixed but the precise values of a_i and b_i may vary depending on $X_1, \ldots, X_{i-1}$. This shows up in the proof of McDiarmid's inequality [McD89], which is described below in §5.3.3.

5.3.3 The method of bounded differences

To use Azuma's inequality, we need a bounded-difference martingale. The easiest way to get such martingales is through the **method of bounded differences**, which was popularized by a survey paper by McDiarmid [McD89]. For this reason the key result is often referred to as **McDiarmid's inequality**.

The basic idea of the method is to structure a problem so that we are computing a function $f(X_1, \ldots, X_n)$ of a sequence of independent random variables $X_1, \ldots, X_n$. To get our martingale, we'll imagine we reveal the X_i one at a time, and compute at each step the expectation of the final value of f based on just the inputs we've seen so far.

Formally, let $\mathcal{F}_t = \langle X_1, \ldots, X_t \rangle$, the σ-algebra generated by X_1 through X_t. This represents all the information we have at time t. Let $Y_t = \mathrm{E}[f \mid \mathcal{F}_t]$, the expected value of f given the values of the first t variables. Then $\{\langle Y_t, \mathcal{F}_T \rangle\}$ forms a martingale, with $Y_0 = \mathrm{E}[f]$ and $Y_n = \mathrm{E}[f \mid X_1, \ldots, X_t] = f$. So if we can find a bound c_t on $|Y_t - Y_{t-1}|$, we can apply Azuma's inequality to get bounds on $Y_n - Y_0 = f - \mathrm{E}[f]$.

A sequence of random variables of the form $Y_t = \mathrm{E}[Z \mid \mathcal{F}_t]$, where $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$ is a called a **Doob martingale**, and this is one of the most common ways to construct a martingale. The proof that a Doob martingale is in fact a martingale is immediate from the general version of the law of iterated expectation $\mathrm{E}[Y_{t+1} \mid \mathcal{F}_t] = \mathrm{E}[\mathrm{E}[Z \mid \mathcal{F}_{t+1}] \mathcal{F}_t \mid =] \mathrm{E}[Z \mid \mathcal{F}_t] = Y_t$. Not all martingales are Doob martingales: for example, the martingale whose difference sequence consists of fair ± 1 coin-flips doesn't converge to any random variable in the limit.[13]

To show that Y_t meets the conditions for Azuma's inequality, we require that f has the **bounded difference property**, which says that there are

[13] A question came up in class whether martingales that converge at some finite time n are technically Doob martingales with respect to $\mathcal{F}_t = \langle Y_0, Y_1, \ldots Y_t \rangle$. This is sort of true, since we have that $\mathrm{E}[Y_n \mid \mathcal{F}_t] = Y_t$ for all t. However this doesn't help much for constructing a particular martingale. If we let $Y_t = 0$ for all $t < k$ and $Y_t = Y_k = \pm 1$ for all $t \geq k$, then for any $n \geq k$, $(\{\langle Y_t, \langle Y_0, \ldots, Y_t \rangle \rangle\})$ is a Doob martingale generated by Y_n. But this doesn't tell us anything about which Y_k does the coin-flip.

bounds c_t such that for any $x_1 \dots x_n$ and any x'_t, we have

$$|f(x_1 \dots x_t \dots x_n) - f(x_1 \dots x'_t \dots x_n)| \leq c_t. \tag{5.3.11}$$

We want to bound $|Y_{t+1} - Y_t| = |\mathrm{E}[f \mid \mathcal{F}_{t+1}] - \mathrm{E}[f \mid \mathcal{F}_t]|$. We can do this showing $Y_{t+1} - Y_t \leq c_{t+1}$. Because f can be replaced by $-f$ without changing the bounded difference property, essentially the same argument will show $Y_{t+1} - Y_t \geq -c_{t+1}$.

Fix some possible value x_{t+1} for X_{t+1}. The bounded difference property says that

$$\begin{aligned} &|f(X_1, \dots, X_t, x_{t+1}, X_{t+2}, \dots, X_n) \\ &\quad - f(X_1, \dots, X_t, X_{t+1}, X_{t+2}, \dots, X_n)| \\ \leq &c_{t+1}, \end{aligned}$$

so

$$\begin{aligned} &|\mathrm{E}[f(X_1, \dots, X_t, x_{t+1}, X_{t+2}, \dots, X_n) \mid X_1, \dots, X_t] \\ &\quad - \mathrm{E}[f(X_1, \dots, X_t, X_{t+1}, X_{t+2}, \dots, X_n) \mid X_1, \dots, X_t]| \\ \leq &c_{t+1}. \end{aligned} \tag{5.3.12}$$

The second conditional expectation is just Y_t. What is the first one? Recall

$$\begin{aligned} Y_{t+1} &= \mathrm{E}[f(X_1, \dots, X_n \mid X_1, \dots, X_{t+1}] \\ &= \mathrm{E}[f(x_1, \dots, x_{t+1}, X_{t+2}, \dots, X_n] \end{aligned}$$

when $X_i = x_i$ for all $i \leq t+1$, since conditioning on $X_1, \dots, X_{t+1}$ just replaces the random variables with their actual values and then averages over the rest. This is the same as

$$\begin{aligned} &\mathrm{E}[f(X_1, \dots, X_t, x_{t+1}, X_{t+2}, \dots, X_n \mid X_1, \dots, X_t] \\ =&\mathrm{E}[f(x_1, \dots, x_{t+1}, X_{t+2}, \dots, X_n)] \end{aligned}$$

when x_{t+1} happens to be the value of X_{t+1}. So the first conditional expectation in (5.3.12) is just Y_{t+1}, giving

$$|Y_{t+1} - Y_t| \leq c_{t+1}.$$

Now we can apply Azuma-Hoeffding to get

$$\Pr[Y_n - Y_0 \geq t] \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^n c_i^2}\right).$$

This turns out to overestimate the possible range of Y_{t+1}. With a more sophisticated argument, it can be shown that for any fixed $x_1, \ldots, x_t$, there exist bounds $a_{t+1} \leq Y_{t+1} - Y_t \leq b_{t+1}$ such that $b_{t+1} - a_{t+1} = c_{t+1}$. We would like to use this to apply the asymmetric version of Azuma-Hoeffding given in (5.3.6). A complication is that the specific values of a_{t+1} and b_{t+1} may depend on the previous values $x_1, \ldots, x_t$, even if the bound c_{t+1} on their maximum difference does not. Fortunately, McDiarmid shows that the inequality works anyway, giving:

Theorem 5.3.4 (**McDiarmid's inequality** [McD89])**.** *Let $X_1, \ldots, X_n$ be independent random variables and let $f(X_1, \ldots, X_n)$ have the bounded difference property with bounds c_i. Then*

$$\Pr\left[f(X_1, \ldots, X_n) - \mathrm{E}\left[f(X_1, \ldots, X_n)\right] \geq t\right] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^n c_i^2}\right). \quad (5.3.13)$$

The main difference between this and a direct application of Azuma-Hoeffding is that the constant factor in the exponent is better by a factor of 4.

Since $-f$ satisfies the same constraints as f, the same bound holds for $\Pr\left[f(X_1, \ldots, X_n) - \mathrm{E}\left[f(X_1, \ldots, X_n)\right] \leq -t\right]$. For some applications it may make sense to apply the union bound to get a two-sided version

$$\Pr\left[\left|f(X_1, \ldots, X_n) - \mathrm{E}\left[f(X_1, \ldots, X_n)\right]\right| \geq t\right] \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^n c_i^2}\right). \quad (5.3.14)$$

5.3.4 Applications

Here are some applications of the preceding inequalities. Most of these are examples of the method of bounded differences.

5.3.4.1 Sprinkling points on a hypercube

Suppose you live in a hypercube, and the local government has conveniently placed mailboxes on some subset A of the nodes. If you start at a random location, how likely is it that your distance to the nearest mailbox deviates substantially from the average distance?

We can describe your position as a bit vector $X_1, \ldots, X_n$, where each X_i is an independent random bit. Let $f(X_1, \ldots, X_n)$ be the distance from $X_1, \ldots, X_n$ to the nearest element of A. Then changing one of the bits

changes this function by at most 1. So we have $\Pr[|f - \mathrm{E}[f]| \geq t] \leq 2e^{-2t^2/n}$ by (5.3.13), giving a range of possible distances that is $O(\sqrt{n \log n})$ with probability at least $1 - n^{-c}$ for any fixed $c > 0$.[14] Of course, without knowing what A is, we don't know what $E[f]$ is; but at least we can be assured that (unless A is very big) the distance we have to walk to send our mail will be pretty much the same pretty much wherever we start.

5.3.4.2 Chromatic number of a random graph

Consider a **random graph** $G(n, p)$ consisting of n vertices, where each possible edge appears with independent probability p. Let χ be the **chromatic number** of this graph, the minimum number of colors necessary if we want to assign a color to each vertex that is distinct for the colors of all of its neighbors. The **vertex exposure martingale** shows us the vertices of the graph one at a time, along with all the edges between vertices that have been exposed so far. We define X_t to be the expected value of χ given this information for vertices $1 \ldots t$.

If Z_i is a random variable describing which edges are present between i and vertices less than i, then the Z_i are all independent, and we can write $\chi = f(Z_1, \ldots, Z_n)$ for some function f (this function may not be very easy to compute, but it exists). Then X_t as defined above is just $\mathrm{E}[f \mid Z_1, \ldots, Z_t]$.

Now observe that f has the bounded difference property with $c_t = 1$: if I change the edges for some vertex v_t, I can't increase the number of colors I need by more than 1, since in the worst case I can always take whatever coloring I previously had for all the other vertices and add a new color for v_t. This implies that the difference between any two graphs G and G' that differ only in the value of some Z_t is at most one, because going between them is an increase in one direction or the other.

McDiarmid's inequality (5.3.13) then says that $\Pr[|\chi - \mathrm{E}[\chi]| \geq t] \leq 2e^{-2t^2/n}$; in other words, the chromatic number of a random graph is tightly concentrated around its mean, even if we don't know what that mean is.

This proof is due to Shamir and Spencer [SS87]. Much better bounds are known on the expected value and distribution of $\chi(G(n, p))$ for many values of n and p than are given by this crude result. For a more recent paper that cites many of these see [Hec21].

[14] Proof: Let $t = \sqrt{\frac{1}{2}(c+1)n \ln n} = O(\sqrt{n \log n})$. Then $2e^{-2t^2/n} = 2e^{-(c+1)\ln n} = 2n^{-c-1} < n^{-c}$ when n is sufficiently large.

5.3.4.3 Balls in bins

Suppose we toss m balls into n bins. How many empty bins do we get? The probability that each bin individually is empty is exactly $(1-1/n)^m$, which is approximately $e^{-m/n}$ when n is large. So the expected number of empty bins is exactly $n(1-1/n)^m$. If we let X_i be the bin that ball i gets tossed into, and let $Y = f(X_1, \ldots, X_m)$ be the number of empty bins, then changing a single X_i can change f by at most 1. So from (5.3.13) we have $\Pr[Y \geq n(1-1/n)^m + t] \leq e^{-2t^2/m}$.

5.3.4.4 Probabilistic recurrence relations

Most probabilistic recurrence arguments (as in Appendix I) can be interpreted as supermartingales: the current estimate of $T(n)$ is always greater than or equal to the expected estimate after doing one stage of the recurrence. This fact can be used to get concentration bounds using (5.3.7).

For example, let's take the recurrence (1.3.1) for the expected number of comparisons for QuickSort:

$$T(n) = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}\left(T(k) + T(n-1-k)\right).$$

We showed in §1.3.1 that the solution to this recurrence satisfies $T(n) \leq 2n \ln n$.

To turn this into a supermartingale, imagine that we carry out a process where we keep around at each step t a set of unsorted blocks of size $n_1^t, n_2^t, \ldots, n_{k_t}^t$ for some k_t (note that the superscripts on n_i^t are not exponents). One step of the process involves choosing one of the blocks (we can do this arbitrarily without affecting the argument) and then splitting that block around a uniformly-chosen pivot. We will track a random variable X_t equal to $C_t + \sum_{i=1}^{k_t} 2n_i^t \ln n_i^t$, where C_t is the number of comparisons done so far and the summation gives an upper bound on the expected number of comparisons remaining.

To show that this is in fact a supermartingale, observe that if we partition a block of size n we add $n-1$ to C_t but replace the cost bound $2n \ln n$ by

an expected

$$\begin{aligned} 2 \cdot \frac{1}{n} \sum_{k=0}^{n-1} 2k \ln k &\leq \frac{4}{n} \int_2^n n \ln n \\ &= \frac{4}{n} \left(\frac{n^2 \ln n}{2} - \frac{n^2}{4} - \ln 2 + 1 \right) \\ &= 2n \ln n - n - \ln 2 + 1 \\ &< 2n \ln n - n. \end{aligned}$$

The net change is less than $-\ln 2$. The fact that it's not zero suggests that we could improve the $2n \ln n$ bound slightly, but since it's going down, we have a supermartingale.

Let's try to get a bound on how much X_t changes at each step. The C_t part goes up by at most $n-1$. The summation can only go down; if we split a block of size n_i, the biggest drop we get is if we split it evenly,[15] This gives a drop of

$$\begin{aligned} 2n \ln n - 2 \left(2 \frac{n-1}{2} \ln \frac{n-1}{2} \right) &= 2n \ln n - 2(n-1) \ln \left(n \frac{n-1}{2n} \right) \\ &= 2n \ln n - 2(n-1)(\ln n - \ln \frac{2n}{n-1}) \\ &= 2n \ln n - 2n \ln n + 2n \ln \frac{2n}{n-1} + 2 \ln n - 2 \ln \frac{2n}{n-1} \\ &= 2n \cdot O(1) + O(\log n) \\ &= O(n). \end{aligned}$$

(with a constant tending to 2 in the limit).

So we can apply (5.3.7) with $c_t = O(n)$ to the at most n steps of the algorithm, and get

$$\Pr\left[C_n - 2n \ln n \geq t\right] \leq e^{-t^2/O(n^3)}.$$

This gives $C_n = O(n^{3/2})$ with constant probability or $O(n^{3/2}\sqrt{\log n})$ with all but polynomial probability. This is a rather terrible bound, but it's much better than $O(n^2)$.

Much tighter bounds are known: QuickSort in fact uses $\Theta(n \log n)$ comparisons with high probability [MH92]. If we aren't too worried about constants, an easy way to see the upper bound side of this is to adapt the

[15] This can be proven most easily using convexity of $n \ln n$.

analysis of Hoare's FIND (§3.6.4). For each element, the number of elements in the same block is multiplied by a factor of at most 3/4 on average each time the element is compared, so the chance that the element is not by itself is at most $(3/4)^k n$ after k comparisons. Setting $k = \log_{4/3}(n^2/\epsilon)$ gives that any particular element is compared k or times with probability at most ϵ/n. The union bound then gives a probability of at most ϵ that the most-compared element is compared k or more times. So the total number of comparisons is $O(\log(n/\epsilon))$ with probability $1 - \epsilon$, which becomes $O(\log n)$ with probability $1 - n^{-c}$ if we set $\epsilon = n^{-c}$ for a fixed c.

We can formalize this argument itself using a supermartingale. Let C_i^t be the number of times i has been compared as a non-pivot in the first t pivot steps and N_i^t be the size of the block containing i after t pivot steps. Let $Y_i^t = (4/3)^{C_i^t} N_i^t$. Then if we pick i's block at step $t+1$, the exponent goes up by at most 1 and N^t drops by a factor of 3/4, canceling out the increase. If we don't pick i's block, Y_i^t is unchanged. In either case we get $Y_i^t \geq \mathrm{E}\left[Y_i^{t+1} \;\middle|\; \mathcal{F}_t\right]$ and $\{Y_i^t\}$ is a supermartingale.

Now let $Z^t = \sum_i Y_i^t$. Since this is greater than or equal to $\sum_i \mathrm{E}\left[Y_i^{t+1} \;\middle|\; \mathcal{F}_t\right] = Z^{t+1}$, $\{Z^t\}$ is also a supermartingale. For $t = 0$, $Z^0 = n \cdot (4/3)^0 \cdot n = n^2$. For $t = n$, $Z^n = \sum_i (4/3)^{C_i^n}$. But then

$$\begin{aligned}
\Pr\left[\max_i C_i^n \geq a \log_{4/3} n\right] &= \Pr\left[\max_i (4/3)^{C_i^n} \geq n^a\right] \\
&\leq \Pr\left[\sum_i (4/3)^{C_i^n} \geq n^a\right] \\
&= \Pr\left[Z^n \geq n^a\right] \\
&\leq \frac{\mathrm{E}\left[Z^n\right]}{n^a} \\
&\leq \frac{Z^0}{n^a} \\
&= n^{a-2}.
\end{aligned}$$

Choosing $a = c+2$ gives an n^{-c} bound on $\Pr\left[\max_i C_i^n \geq (c+2)\log_{4/3} n\right]$ and thus the same bound on $\Pr\left[\sum_i C_i^n \geq (c+2) n \log_{4/3} n\right]$. This shows that the total number of comparisons is $O(n \log n)$ with high probability.

5.3.4.5 Multi-armed bandits

In the **multi-armed bandit** problem, we must choose at each time step one of a fixed set of k **arms** to pull. Pulling arm i at time t yields a return

of X_i^t, a random payoff typically assumed to be between 0 and 1. Suppose that all the X_i^t are independent, and that for each fixed i, all X_i^t have the same distribution, and thus the same expected payoff. Suppose also that we initially know nothing about these distributions. What strategy can we use to maximize our expected payoff over a large number of pulls?

More specifically, we want to minimize our **regret**, defined as

$$T_i \cdot (\mu^* - \mu_i), \tag{5.3.15}$$

where T_i counts the number of times we pull arm i, where μ^* is the expected payoff of the best arm, and $\mu_i = \mathrm{E}[X_i]$ is the expected payoff of arm i.

The tricky part here is that when we pull an arm and get a bad return, we don't know if we were just unlucky this time or it's actually a bad arm. So we have an incentive to try lots of different arms. On the other hand, the more we pull a genuinely inferior arm, the worse our overall return. We'd like to adopt a strategy that trades off between exploration (trying new arms) and exploitation (collecting on the best arm so far) to do as best we can in comparison to a strategy that always pulls the best arm.

The UCB1 algorithm Fortunately, there is a simple algorithm due to Auer *et al.* [ACBF02] that solves this problem for us.[16] To start with, we pull each arm once. For any subsequent pull, suppose that for each i, we have pulled the i-th arm n_i times so far. Let $n = \sum n_i$, and let $\overline{x}_i$ be the average payoff from arm i so far. Then the **UCB1** algorithm pulls the arm that maximizes

$$\overline{x}_i + \sqrt{\frac{2 \ln n}{n_i}}. \tag{5.3.16}$$

UCB stands for **upper confidence bound**. (The "1" is because it is the first, and simplest, of several algorithms of this general structure given in the paper.) The idea is that we give arms that we haven't tried very much the benefit of the doubt, and assume that their actual average payout lies at the upper end of some plausible range of values.[17]

[16] This is not the only algorithm for solving multi-armed bandit problems, and it's not even the only algorithm in the Auer *et al.* paper. But it has the advantage of being relatively straightforward to analyze. For a more general survey of multi-armed bandit algorithms, see [BCB12] or [LS20].

[17] The "bound" part is because we don't attempt to compute the exact upper end of this confidence interval, which may be difficult, but instead use an upper bound derived from Hoeffding's inequality. This distinguishes the UCB1 algorithm of [ACBF02] from the *upper confidence interval* approach of Lai and Robbins [LR85] that it builds on.

The quantity $\sqrt{\frac{2 \ln n}{n_i}}$ is a bit mysterious, but it arises in a fairly natural way from the asymmetric version of Hoeffding's inequality. With a small adjustment to deal with non-zero-mean variables, (5.3.4) says that, if S is a sum of n random variables bounded between a_i and b_i, then

$$\Pr\left[\sum_{i=1}^{n}(X_i - \mathrm{E}\left[X_i\right]) \geq t\right] \leq e^{-2t^2/\sum_{i=1}^{n}(b_i-a_i)^2}. \tag{5.3.17}$$

By applying (5.3.4) to $-X_i$, we also get

$$\Pr\left[\sum_{i=1}^{n}(X_i - \mathrm{E}\left[X_i\right]) \leq -t\right] \leq e^{-2t^2/\sum_{i=1}^{n}(b_i-a_i)^2}. \tag{5.3.18}$$

Now consider $\overline{x}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} X_j$ where each X_j lies between 0 and 1. This is equivalent to having $\overline{x}_i = \sum_{j=1}^{n_i} Y_j$ where $Y_j = X_j/n_i$ lies between 0 and $1/n_i$. So (5.3.17) says that

$$\begin{aligned}\Pr\left[\overline{x}_i - \mathrm{E}\left[\overline{x}_i\right] \geq \sqrt{\frac{2 \ln n}{n_i}}\right] &\leq e^{-2(\sqrt{(2 \ln n)/n_i})^2/(n_i(1/n_i)^2)} \\ &= e^{-4 \ln n} \\ &= n^{-4}.\end{aligned} \tag{5.3.19}$$

We also get a similar lower bound using (5.3.18).

So the bonus applied to $\overline{x}_i$ is really a high-probability bound on how big the difference between the observed payoff and expected payoff might be. The $\sqrt{\ln n}$ part is there to make the error probability be small as a function of n, since we will be summing over a number of bad cases polynomial in n and not a particular n_i. Applying the bonus to all arms make it likely that the observed payoff of the best arm stays above its actual payoff, so we won't forget to pull it. The hope is that over time the same bonus applied to other arms will not boost them up so much that we pull them any more than we have to.

However, in an infinite execution of the algorithm, even a bad arm will be pulled infinitely often, as $\ln n$ rises enough to compensate for the last increase in n_i. This accounts for an $O(\log n)$ term in the regret, as we will see below. It also prevents us from getting stuck refusing to give a good arm a second chance just because we had an initial run of bad luck.

Analysis of UCB1 The following theorem is a direct quote of [ACBF02, Theorem 1]:

Theorem 5.3.5 ([ACBF02]). *For all $K > 1$, if policy UCB1 is run on K machines having arbitrary reward distributions $P_1, \ldots, P_K$ with support in $[0, 1]$, then its expected regret after any number n of plays is at most*

$$\left[8 \sum_{i:\mu_i<\mu^*} \frac{\ln n}{\Delta_i}\right] + \left(1 + \frac{\pi^2}{3}\right)\left(\sum_{j=1}^{K} \Delta_j\right). \tag{5.3.20}$$

Here $\mu_i = \mathrm{E}[P_i]$ is the expected payoff for arm i, μ^* as before is $\max_i \mu_i$, and $\Delta_i = \mu^* - \mu_i$ is the regret for pulling arm i. The theorem states that our expected regret is bounded by a term for each arm worse that μ^* that grows logarithmically in n and is inversely proportional to how close the arm is to optimal, plus a constant additional loss corresponding to pulling every arm a little more than 4 times on average. The logarithmic regret in n is a bit of a nuisance, but an earlier lower bound of Lai and Robbins [LR85] shows that something like this is necessary in the limit.

To prove Theorem 5.3.5, we need to get an upper bound on the number of times each suboptimal arm is pulled during the first n pulls. Define

$$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}, \tag{5.3.21}$$

the bonus given to an arm that has been pulled s times in the first t pulls.

Fix some optimal arm. Let $\overline{X}_{i,s}$ be the average return on arm i after s pulls and $\overline{X}^*_s$ be the average return on the optimal arm after s pulls.

If we pull arm i after t total pulls, when arm i has previously been pulled s_i times and our optimal arm has been pulled s^* times, then we must have

$$\overline{X}_{i,s_i} + c_{t,s_i} \geq \overline{X}^*_{s^*} + C_{t,s^*}. \tag{5.3.22}$$

This just says that arm i with its bonus looks better than the optimal arm with its bonus.

To show that this bad event doesn't happen, we need three things:

1. The value of $\overline{X}^*_{s^*} + c_{t,s^*}$ should be at least μ^*. Were it to be smaller, the observed value $\overline{X}^*_{s^*}$ would be more than c_{t,s^*} away from its expectation. Hoeffding's inequality implies this doesn't happen too often.

2. The value of $\overline{X}_{i,s_i} + c_{t,s_i}$ should not be too much bigger than μ_i. We'll again use Hoeffding's inequality to show that $\overline{X}_{i,s_i}$ is likely to be at most $\mu_i + c_{t,s_i}$, making $\overline{X}_{i,s_i} + c_{t,s_i}$ at most $\mu_i + 2c_{t,s_i}$.

3. The bonus c_{t,s_i} should be small enough that even adding $2c_{t,s_i}$ to μ_i is not enough to beat μ^*. This means that we need to pick s_i large enough that $2c_{t,s_i} \leq \Delta_i$. For smaller values of s_i, we will just accept that we need to pull arm i a few more times before we wise up.

More formally, if none of the following events hold:

$$\overline{X}^*_{s^*} + c_{t,s^*} \leq \mu^*. \tag{5.3.23}$$

$$\overline{X}_{i,s_i} \geq \mu_i + c_{t,s_i} \tag{5.3.24}$$

$$\mu^* - \mu_i < 2c_{t,s_i}, \tag{5.3.25}$$

then $\overline{X}^*_{s^*} + c_{t,s^*} > \mu^* > \mu_i + 2c_{t,s_i} > \overline{X}_{i,s_i} + c_{t,s_i}$, and we don't pull arm i because the optimal arm is better. (We don't necessarily pull the optimal arm, but if we don't, it's because we pull some other arm that still isn't arm i.)

For (5.3.23) and (5.3.24), we repeat the argument in (5.3.19), plugging in t for n and s_i or s^* for n_i. This gives a probability of at most $2t^{-4}$ that either or both of these bad events occur.

For (5.3.25), we need to do something a little sneaky, because the statement is not actually true when s_i is small. So we will give ℓ_i free pulls to arm i, and only start comparing arm i to the optimal arm after we have done at least this many pulls. The value of ℓ_i is chosen so that, when $t \leq n$ and $s_i > \ell_i$,

$$2c_{t,s_i} \leq \mu^* - \mu_i,$$

which expands to,

$$2\sqrt{\frac{2\ln t}{s_i}} \leq \Delta_i,$$

giving

$$s_i \geq \frac{8\ln t}{\Delta_i^2}.$$

So we must set ℓ_i to be at least

$$\frac{8\ln n}{\Delta_i^2} \geq \frac{8\ln t}{\Delta_i^2}.$$

Because ℓ_i must be an integer, we actually get

$$\ell_i = \left\lceil \frac{8\ln n}{\Delta_i^2} \right\rceil \leq 1 + \frac{8\ln n}{\Delta_i^2}.$$

This explains (after multiplying by the regret Δ_i) the first term in (5.3.20).

For the other sources of bad pulls, apply the union bound to the $2t^{-4}$ error probabilities we previously computed for all choices of $t \leq n$, $s^* \geq 1$, and $s_i > \ell_i$. This gives

$$\begin{aligned}\sum_{t=1}^{n} \sum_{s^*=1}^{t-1} \sum_{s_i=\ell_i+1}^{t-1} 2t^{-4} &< 2\sum_{t=1}^{\infty} t^2 \cdot t^{-4} \\ &= 2 \cdot \frac{\pi^2}{6} \\ &= \frac{\pi^2}{3}.\end{aligned}$$

Again we have to multiply by the regret Δ_i for pulling the i-th arm, which gives the second term in (5.3.20).

5.4 Relation to limit theorems

Since in many cases we are working with sums $S_n = \sum_{i=1}^{n} X_i$ of independent, identically distributed random variables X_i, classical limit theorems apply. These relate the behavior of S_n in the limit to the common expectation $\mu = \mathrm{E}[X_i]$ and variance $\sigma^2 = \mathrm{Var}[X_i]$.[18]

These include the **strong law of large numbers**

$$\Pr \lim_{n\to\infty} S_n/n = \mu = 1 \tag{5.4.1}$$

and the **central limit theorem** (**CLT**)

$$\lim_{n\to\infty} \Pr\left[\frac{S_n - \mu n}{\sigma\sqrt{n}} \leq t\right] = \Phi(t), \tag{5.4.2}$$

where Φ is the normal distribution function.

These can sometimes be useful in the analysis if randomized algorithms, but often are not strong enough to get the results we want. The main problem is that the standard versions of both the strong law and the central limit theorem say nothing about rate of convergence. So if we want (for example) to use the CLT to show that S_n is exponentially unlikely to be $n\sigma$ away from the mean, we can't do it directly, because $n\sigma/\sigma$ is not a fixed constant t,

[18]The quantity $\sigma = \sqrt{\mathrm{Var}[X]}$ is called the **standard deviation** of X, and informally gives a measure of the typical distance of X from its mean, measured in the same units as X.

and for any fixed constant t, we don't know when the limit behavior actually starts working.

But there are variants of these theorems that do bound rate of convergence that can be useful in some cases. An example is given in §5.5.1.

5.5 Anti-concentration bounds

It may be that for some problem you want to show that a sum of random variables is far from its mean at least some of the time: this would be an **anti-concentration bound**. Anti-concentration bounds are much less well-understood than concentration bounds, but there are known results that can help in some cases.

For variables where we know the distribution of the sum exactly (e.g., sums with binomial distributions, or sums we can attack with generating functions), we don't need these. But they may be useful if computing the distribution of the sum directly is hard.

5.5.1 The Berry-Esseen theorem

The **Berry-Esseen theorem**[19] is an extension of the central limit theorem that characterizes how quickly a sum of independent identically-distributed random variables converges to a normal distribution, as a function of the **third moment** of the random variables. Its simplest version says that if we have n independent, identically-distributed random variables $X_1 \dots X_n$, with $\mathrm{E}[X_i] = 0$, $\mathrm{Var}[X_i] = \mathrm{E}[X_i^2] = \sigma^2$, and $\mathrm{E}\left[|X_i|^3\right] \leq \rho$, then

$$\sup_{-\infty<x<\infty} \left| \Pr\left[\frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i \leq x\right] - \Phi(x) \right| \leq \frac{C\rho}{\sigma^3\sqrt{n}}, \tag{5.5.1}$$

were C is an absolute constant and Φ is the normal distribution function. Note that the σ^3 in the denominator is really $\mathrm{Var}[X_i]^{3/2}$. Since the probability bound doesn't depend on x, it's more useful toward the middle of the distribution than in the tails.

A classic proof of this result with $C = 3$ can be found in [Fel71, §XVI.5]. Shevtsova [She11] shows a stronger bound of $C < 0.4784$.

As with most results involving sums of random variables, there are generalizations to martingales. These are too involved to describe here, but see [HH80, §3.6].

[19] Sometimes written *Berry-Esséen theorem* to help with the pronunciation of Esseen's last name.

5.5.2 The Littlewood-Offord problem

The **Littlewood-Offord problem** asks, given a set of n complex numbers $x_1 \ldots x_n$ with $|x_i| \geq 1$, for how many assignments of ± 1 to coefficients $\epsilon_1 \ldots \epsilon_n$ it holds that $|\sum_{i=1}^{n} \epsilon_i x_i| \leq r$. Paul Erdős showed [Erd45] that this quantity was at most $cr2^n/\sqrt{n}$, where c is a constant. The quantity $c2^n/\sqrt{n}$ here is really $\frac{1}{2}\binom{n}{\lfloor n/2 \rfloor}$: Erdős's proof shows that for each interval of length $2r$, the number of assignments that give a sum in the interior of the interval is bounded by at most the sum of the r largest binomial coefficients.

In random-variable terms, this means that if we are looking at $\sum_{i=1}^{n} \epsilon x_i$, where the x_i are constants with $|x_i| \geq 1$ and the ϵ_i are independent ± 1 fair coin-flips, then $\Pr\left[|\sum_{i=1}^{n} \epsilon_i x_i| \leq r\right]$ is maximized by making all the x_i equal to 1. This shows that any distribution where the x_i are all reasonably large will not be any more concentrated than a binomial distribution.

There has been a lot of more recent work on variants of the Littlewood-Offord problem, much of it by Terry Tao and Van Vu. See http://terrytao.wordpress.com/2009/02/16/a-sharp-inverse-littlewood-offord-theorem/ for a summary of much of this work.

Chapter 6

Randomized search trees

These are data structures that are either trees or equivalent to trees, and use randomization to maintain balance. We'll start by reviewing deterministic binary search trees and then add in the randomization.

6.1 Binary search trees

A **binary search tree** is a standard data structure for holding sorted data. A **binary tree** is either empty, or it consists of a **root** node containing a **key** and pointers to left and right **subtrees**. What makes a binary tree a binary search tree is the invariant that, both for the tree as a whole and any subtree, all keys in the left subtree are less than the key in the root, while all keys in the right subtree are greater than the key in the root. This ordering property means that we can search for a particular key by doing binary search: if the key is not at the root, we can recurse into the left or right subtree depending on whether it is smaller or bigger than the key at the root.

The efficiency of this operation depends on the tree being **balanced**. If each subtree always holds a constant fraction of the nodes in the tree, then each recursive step throws away a constant fraction of the remaining nodes. So after $O(\log n)$ steps, we find the key we are looking for (or find that the key is not in the tree). But the definition of a binary search tree does not by itself guarantee balance, and in the worst case a binary search tree degenerates into a linked list with $O(n)$ cost for all operations (see Figure 6.2.

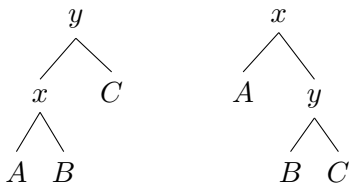


Figure 6.1: Tree rotations

6.1.1 Rebalancing and rotations

Deterministic binary search tree implementations include sophisticated rebalancing mechanisms to adjust the structure of the tree to preserve balance as nodes are inserted or delete. Typically this is done using **rotations**, which are operations that change the position of a parent and a child while preserving the left-to-right ordering of keys (see Figure 6.1).

Examples include **AVL trees** [AVL62], where the left and right subtrees of any node have heights that differ by at most 1; **red-black trees** [GS78], where a coloring scheme is used to maintain balance; and **scapegoat trees** [GR93], where no information is stored at a node but part of the tree is rebuilt from scratch whenever an operation takes too long. These all give $O(\log n)$ cost per operation (amortized in the case of scapegoat trees), and vary in how much work is needed in rebalancing. Both AVL trees and red-black trees perform more rotations than randomized rebalancing does on average.

6.2 Random insertions

Suppose we insert n keys into an initially-empty binary search tree in random order with no rebalancing. This means that for each insertion, we follow the same path that we would when searching for the key, and when we reach an empty tree, we replace it with a tree consisting solely of the key at the root.[1]

Since we chose a random order, each element is equally likely to be the root, and all the elements less than the root end up in the left subtree, while all the elements greater than the root end up in the right subtree, where they are further partitioned recursively. This is exactly what happens in randomized QuickSort (see §1.3.1), so the structure of the tree will exactly

[1]This is not the only way to generate a binary search tree at random. For example, we could instead choose uniformly from the set of all C_n binary search trees with n nodes, where $C_n = \frac{1}{n+1}\binom{2n}{n}$ is the n-th **Catalan number**. For $n \geq 3$, this gives a different distribution that we don't care about.

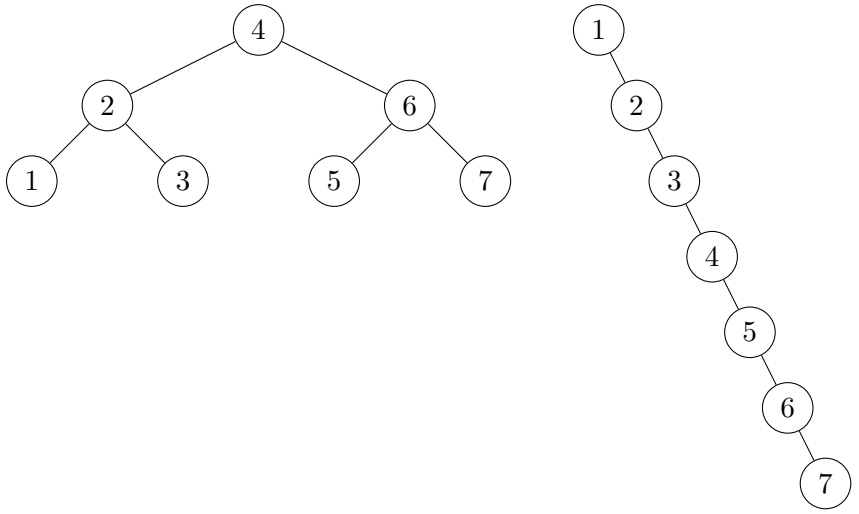


Figure 6.2: Balanced and unbalanced binary search trees

mirror the structure of an execution of QuickSort. So, for example, we can immediately observe from our previous analysis of QuickSort that the **total path length**—the sum of the depths of the nodes—is $\Theta(n \log n)$, since the depth of each node is equal to 1 plus the number of comparisons it participates in as a non-pivot, and (using the same argument as for Hoare's FIND in §3.6.4) that the height of the tree is $O(\log n)$ with high probability.[2]

When n is small, randomized binary search trees can look pretty scraggly. Figure 6.3 shows a typical example.

The problem with this approach in general is that we don't have any guarantees that the input will be supplied in random order, and in the worst case we end up with a linked list, giving $O(n)$ worst-case cost for all operations.

[2]The argument for Hoare's FIND is that any node has at most 3/4 of the descendants of its parent on average; this gives for any node x that $\Pr[\text{depth}(x) > d] \leq (3/4)^{d-1}n$, or a probability of at most n^{-c} that $\text{depth}(x) > 1 + (c+1)\log(n)/\log(4/3) \approx 1 + 6.952 \ln n$ for $c = 1$, which we need to apply the union bound. The right answer for the actual height of a randomly-generated search tree in the limit is $4.31107 \ln n$ [Dev88] so this bound is actually pretty close. The real height is still nearly a factor of three worse than for a completely balanced tree, which has max depth bounded by $1 + \lg n \approx 1 + 1.44269 \ln n$.

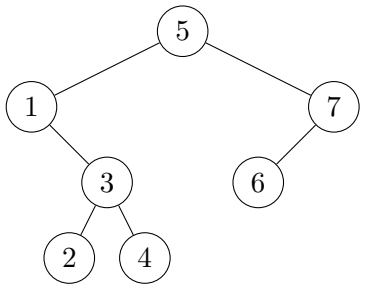


Figure 6.3: Binary search tree after inserting 5 1 7 3 4 6 2

6.3 Treaps

The solution to bad inputs is the same as for QuickSort: instead of assuming that the input is permuted randomly, we assign random priorities to each element and organize the tree so that elements with higher priorities rise to the top. The resulting structure is known as a **treap** [SA96], because it satisfies the binary search tree property with respect to keys and the **heap** property with respect to priorities.[3]

There's an extensive page of information on treaps at `http://faculty.washington.edu/aragon/treaps.html`, maintained by Cecilia Aragon, the co-inventor of treaps; they are also discussed at length in [MR95, §8.2]. We'll give a brief description here.

To insert a new node in a treap, first walk down the tree according to the key and insert the node as a new leaf. Then go back up fixing the heap property by rotating the new element up until it reaches an ancestor with the same or higher priority. (See Figure 6.4 for an example.) Deletion is the reverse of insertion: rotate a node until it has 0 or 1 children (by swapping with its higher-priority child at each step), and then prune it out, connecting its child, if any, directly to its parent.

Because of the heap property, the root of each subtree is always the element in that subtree with the highest priority. This means that the structure of a treap is completely determined by the priorities and the keys, no matter what order the elements arrive in. We can imagine in retrospect

[3]The name "treap" for this data structure is now standard but the history is a little tricky. According to Seidel and Aragon, essentially the same data structure (though with non-random priorities) was previously called a **cartesian tree** by Vuillemin [Vui80], and the word "treap" was initially applied by McCreight to a different data structure—designed for storing two-dimensional data—that was called a **priority search tree** in its published form. [McC85].

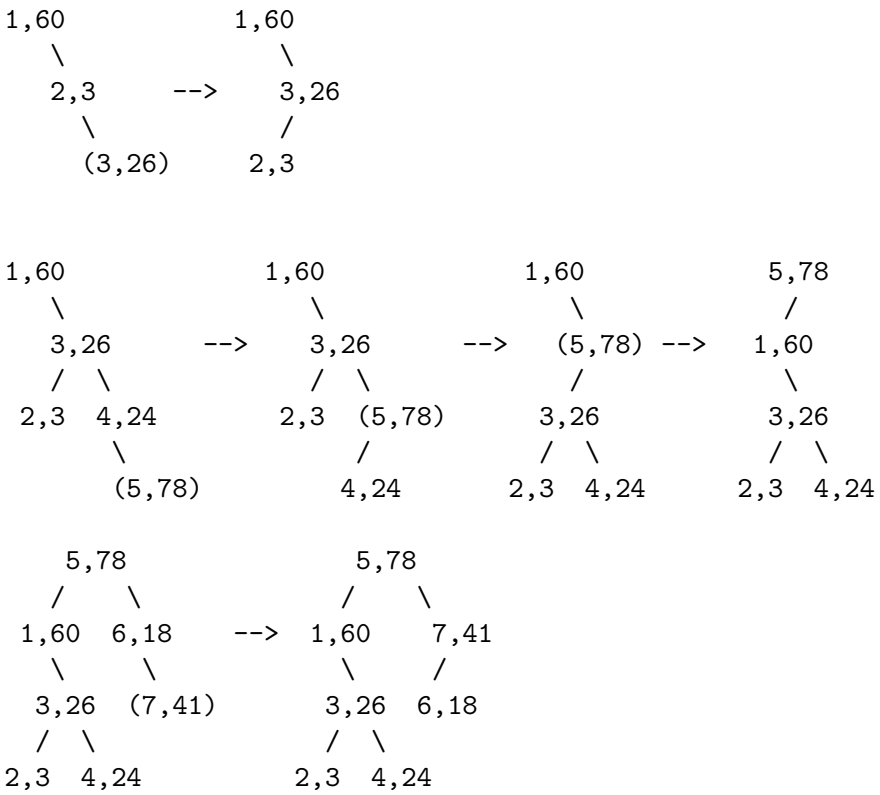


Figure 6.4: Inserting values into a treap. Each node is labeled with k, p where k is the key and p the priority. Insertion of values not requiring rotations are not shown.

that the treap is constructed recursively by choosing the highest-priority element as the root, then organizing all smaller-index and all larger-index nodes into the left and right subtrees by the same rule.

If we assign the priorities independently and uniformly at random from a sufficiently large set ($\omega(n^2)$ is enough in the limit), then we get no duplicates, and by symmetry all $n!$ orderings are equally likely. So the analysis of the depth of a treap with random priorities is identical to the analysis of a binary search tree with random insertion order. It's not hard to see that the costs of search insertion, and deletion operations are all linear in the depth of the tree, so the expected cost of each of these operations is $O(\log n)$.

6.3.1 Assumption of an oblivious adversary

One caveat is that this only works if the priorities of the elements of the tree are in fact independent. If operations on the tree are chosen by an **adaptive adversary**, this assumption may not work. An adaptive adversary is one that can observe the choice made by the algorithm and react to them: in this case, a simple strategy would be to insert elements 1, 2, 3, 4, etc., in order, deleting each one and reinserting it until it has a lower priority value than all the smaller elements. This might take a while for the later elements, but the end result is the linked list again. For this reason it is standard to assume in randomized data structures that the adversary is **oblivious**, meaning that it has to specify a sequence of operations without knowing what choices are made by the algorithm. Under this assumption, whatever insert or delete operations the adversary chooses, at the end of any particular sequence of operations we still have independent priorities on all the remaining elements, and the $O(\log n)$ analysis goes through.

6.3.2 Analysis

The analysis of treaps as carried out by Seidel and Aragon [SA96] is a nice example of how to decompose a messy process into simple variables, much like the linearity-of-expectation argument for QuickSort (§1.3.2). The key observation is that it's possible to bound both the expected depth of any node and the number of rotations needed for an insert or delete operation directly from information about the ancestor-descendant relationship between nodes.

Define two classes of indicator variables. For simplicity, we assume that the elements have keys 1 through n, which we also use as indices.

1. $A_{i,j}$ indicates the event that i is an **ancestor** of j, where i is an ancestor of j if it appears on the path from the root to j. Note that every node

is an ancestor of itself.

2. $C_{i;\ell,m}$ indicates the event that i is a **common ancestor** of both ℓ and m; formally, $C_{i;\ell,m} = A_{i,\ell}A_{i,m}$.

The nice thing about these indicator variables is that it's easy to compute their expectations.

For $A_{i,j}$, i will be the ancestor of j if and only if i has a higher priority than j and there is no k between i and j that has an even higher priority: in other words, if i has the highest priority of all keys in the interval $[\min(i,j), \max(i,j)]$. To see this, imagine that we are constructing the treap recursively, by starting with all elements in a single interval and partitioning each interval by its highest-priority element. Consider the last interval in this process that contains both i and j, and suppose $i < j$ (the $j > i$ case is symmetric). If the highest-priority element is some k with $i < k < j$, then i and j are separated into distinct intervals and neither is the ancestor of the other. If the highest-priority element is j, then j becomes the ancestor of i. The highest-priority element can't be less than i or greater than j, because then we get a smaller interval that contains both i and j. So the only case where i becomes an ancestor of j is when i has the highest priority.

It follows that $\mathrm{E}[A_{i,j}] = \frac{1}{|i-j|+1}$, where the denominator is just the number of elements in the range $[\min(i,j), \max(i,j)]$.

For $C_{i;\ell,m}$, i is the common ancestor of both ℓ and m if and only if it is has the highest priority in both $[\min(i,\ell), \max(i,\ell)]$ and $[\min(i,m), \max(i,m)]$. It turns out that no matter what order i, ℓ, and m come in, these intervals overlap so that i must have the highest priority in $[\min(i,\ell,m), \max(i,\ell,m)]$. This gives $\mathrm{E}[C_{i;\ell,m}] = \frac{1}{\max(i,\ell,m)-\min(i,\ell,m)+1}$.

6.3.2.1 Searches

From the $A_{i,j}$ variables we can compute $\text{depth}(j) = \sum_i A_{i,j} - 1$.[4] So

$$\begin{aligned}
\mathrm{E}\left[\text{depth}(j)\right] &= \left(\sum_{i=1}^{n} \frac{1}{|i-j|+1}\right) - 1 \\
&= \left(\sum_{i=1}^{j} \frac{1}{j-i+1}\right) + \left(\sum_{i=j+1}^{n} \frac{1}{i-j+1}\right) - 1 \\
&= \left(\sum_{k=1}^{j} \frac{1}{k}\right) + \left(\sum_{k=2}^{n-j+1} \frac{1}{k}\right) - 1 \\
&= H_j + H_{n-j+1} - 2.
\end{aligned}$$

This is maximized at $j = (n+1)/2$, giving $2H_{(n+1)/2} - 2 = 2\ln n + O(1)$. So we get the same $2\ln n + O(1)$ bound on the expected depth of any one node that we got for QuickSort. We can also sum over all j to get the exact value of the expected total path length (but we won't). These quantities bound the expected cost of searches.

6.3.2.2 Insertions and deletions

For insertions and deletions, the question is how many rotations we have to perform to float a new leaf up to its proper location (after an insertion) or to float a deleted node down to a leaf (before a deletion). Since insertion is just the reverse of deletion, we can get a bound on both by concentrating on deletion. The trick is to find some metric for each node that (a) bounds the number of rotations needed to move a node to the bottom of the tree and (b) is easy to compute based on the A and C variables

The **left spine** of a subtree is the set of all nodes obtained by starting at the root and following left pointers; similarly the **right spine** is what we get if we follow the right pointers instead.

When we rotate an element down, we are rotating either its left or right child up. This removes one element from either the right spine of the left subtree or the left spine of the right subtree, but the rest of the spines are left intact (see Figure 6.5). Subsequent rotations will eventually remove all these elements by rotating them above the target, while other elements in

[4] We need the -1 because of the convention that the root has depth 0, making the depth of a node one less than the number of its ancestors. Equivalently, we could exclude j from the sum and count only proper ancestors.

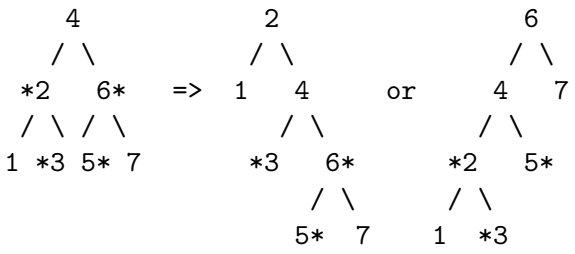


Figure 6.5: Rotating 4 right shortens the right spine of its left subtree by removing 2; rotating left shortens the left spine of the right subtree by removing 6.

the subtree will be carried out from under the target without ever appearing as a child or parent of the target. Because each rotation removes exactly one element from one or the other of the two spines, and we finish when both are empty, the sum of the length of the spines gives the number of rotations.

To calculate the length of the right spine of the left subtree of some element ℓ, start with the predecessor $\ell - 1$ of ℓ. Because there is no element between them, either $\ell - 1$ is a descendant of ℓ or an ancestor of ℓ. In the former case (for example, when ℓ is 4 in Figure 6.5), we want to include all ancestors of $\ell - 1$ up to ℓ itself. Starting with $\sum_i A_{i,\ell-1}$ gets all the ancestors of $\ell - 1$, and subtracting off $\sum_i C_{i;\ell-1,\ell}$ removes any common ancestors of $\ell - 1$ and ℓ. Alternatively, if $\ell - 1$ is an ancestor of ℓ, every ancestor of $\ell - 1$ is also an ancestor of ℓ, so the same expression $\sum_i A_{i,\ell-1} - \sum_i C_{i;\ell-1,\ell}$ evaluates to zero.

It follows that the expected length of the right spine of the left subtree is exactly

$$\begin{aligned}
\mathrm{E}\left[\sum_{i=1}^{n} A_{i,\ell-1} - \sum_{i=1}^{n} C_{i;\ell-1,\ell}\right] &= \sum_{i=1}^{n} \frac{1}{|i-(\ell-1)|+1} - \sum_{i=1}^{n} \frac{1}{\max(i,\ell) - \min(i,\ell-1)+1} \\
&= \sum_{i=1}^{\ell-1} \frac{1}{\ell - i} + \sum_{i=\ell}^{n} \frac{1}{i-\ell+2} - \sum_{i=1}^{\ell-1} \frac{1}{\ell-i+1} - \sum_{i=\ell}^{n} \frac{1}{i-(\ell-1)+1} \\
&= \sum_{j=1}^{\ell-1} \frac{1}{j} + \sum_{j=2}^{n-\ell+2} \frac{1}{j} - \sum_{j=2}^{\ell} \frac{1}{j} - \sum_{j=2}^{n-\ell+2} \frac{1}{j} \\
&= 1 - \frac{1}{\ell}.
\end{aligned}$$

By symmetry, the expected length of the left spine of the right subtree is $1 - \frac{1}{n-\ell+1}$. So the total expected number of rotations needed to delete the

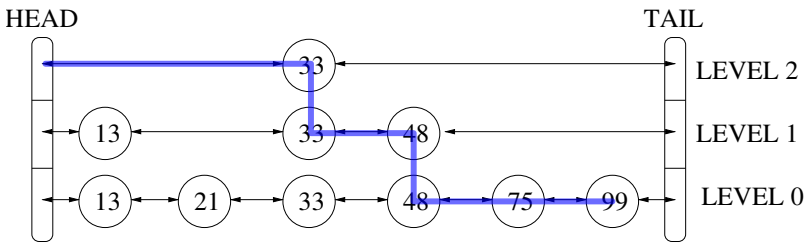


Figure 6.6: A skip list. The blue search path for 99 is superimposed on an original image from [AS07].

ℓ-th element is

$$2 - \frac{1}{\ell} - \frac{1}{n - \ell + 1} \leq 2.$$

6.3.2.3 Other operations

Treaps support some other useful operations: for example, we can split a treap into two treaps consisting of all elements less than and all elements greater than a chosen pivot by rotating the pivot to the root ($O(\log n)$ rotations on average, equal to the pivot's expected depth as calculated in §6.3.2.1) and splitting off the left and right subtrees. Merging two treaps with non-overlapping keys is the reverse of this and so it has the same expected complexity.

6.4 Skip lists

A skip list [Pug90] is a randomized tree-like data structure based on linked lists. It consists of a level 0 list that is an ordinary sorted linked list, together with higher-level lists that contain a random sampling of the elements at lower levels. When inserted into the level i list, an element flips a coin that tells it with probability p to insert itself in the level $i + 1$ list as well. The result is that the element is represented by a tower of nodes, one in each of the bottom $1 + X$ many layers, where X is a geometrically-distributed random variable. An example of a small skip list is shown in Figure 6.6.

Searches in a skip list are done by starting in the highest-level list and searching forward for the last node whose key is smaller than the target; the search then continues in the same way on the next level down. To bound the expected running time of a search, it helps to look at this process backwards; the reversed search path starts at level 0 and continues going backwards

until it reaches the first element that is also in a higher level; it then jumps to the next level up and repeats the process. The nice thing about this reversed process is that it has a simple recursive structure: if we restrict a skip list to only those nodes to the left of and at the same level or higher of a particular node, we again get a skip list. Furthermore, the structure of this restricted skip list depends only on coin-flips taken at nodes within it, so it's independent of anything that happens elsewhere in the full skip list.

We can analyze this process by tracking the number of nodes in the restricted skip list described above, which is just the number of nodes in the current level that are earlier than the current node. If we move left, this drops by 1; if up, this drops to p times its previous value on average. So the number of such nodes X_k after k steps satisfies the probabilistic recurrence

$$\begin{aligned}\mathrm{E}\left[X_{k+1} \mid X_k\right] &= (1-p)(X_k - 1) + p(pX_k) \\ &= (1 - p + p^2)X_k - (1-p) \\ &\le (1 - p + p^2)X_k,\end{aligned}$$

with $X_0 = n - 1$ (since the last node is not included). Wrapping expectations around both sides gives $\mathrm{E}\left[X_{k+1}\right] = \mathrm{E}\left[\mathrm{E}\left[X_{k+1} \mid X_k\right]\right] \le (1 - p + p^2)\,\mathrm{E}\left[X_k\right]$, and in general we get $\mathrm{E}\left[X_k\right] \le (1 - p + p^2)^k X_0 < (1 - p + p^2)^k n$. This is minimized at $p = 1/2$, giving $\mathrm{E}\left[X_k\right] \le (3/4)^k n$, suspiciously similar to the bound we computed before for random binary search trees.

When $X_k = 0$, our search is done, so if T is the time to search, we have $\Pr\left[T \ge k\right] = \Pr\left[X_k \ge 1\right] \le (3/4)^k n$, by Markov's inequality. In particular, if we want to guarantee that we finish with probability $1 - \epsilon$, we need to run for $\log_{4/3}(n/\epsilon)$ steps. This translates into an $O(\log n)$ bound on the expected search time, and the constant is even the same as our (somewhat loose) bound for treaps.

The space per element of a skip list also depends on p. Every element needs one pointer for each level it appears in. The number of levels each element appears in is a geometric random variable where we are waiting for an event of probability $1 - p$, so the expected number of pointers is $\frac{1}{1-p}$. For constant p this is $O(1)$. However, the space cost can reduced (at the cost of increasing search time) by adjusting p. For example, if space is at a premium, setting $p = 1/10$ produces $10/9$ pointers per node on average—not much more than in a linked list—but still gives $O(\log n)$ search time.

In general the trade-off is between $n\left(\frac{1}{1-p}\right)$ total expected pointers and $\log_{1/(1-p+p^2)}(n/\epsilon)$ search time. For small p, the number of pointers scales as $1 + O(p)$, while the constant factor in the search time is $\frac{1}{-\log(1-p+p^2)} = O(1/p)$.

Skip lists are even easier to split or merge than treaps. It's enough to cut (or recreate) all the pointers crossing the boundary, without changing the structure of the rest of the list.

Chapter 7

Hashing

The basic idea of hashing is that we have keys from a large set U, and we'd like to pack them in a small set M by passing them through some function $h : U \to M$, without getting too many **collisions**, pairs of distinct keys x and y with $h(x) = h(y)$. Where randomization comes in is that we want this to be true even if the adversary picks the keys to hash. At one extreme, we could use a random function, but this will take a lot of space to store.[1] So our goal will be to find functions with succinct descriptions that are still random enough to do what we want.

The presentation in this chapter is based largely on [MR95, §§8.4-8.5] (which is in turn based on work of Carter and Wegman [CW77] on universal hashing and Fredman, Komlós, and Szemerédi [FKS84] on $O(1)$ worst-case hashing); on [PR04] and [Pag06] for cuckoo hashing; and [MU17, §5.5.3] for Bloom filters.

7.1 Hash tables

Here we review **hash tables**, which are implementations of the **dictionary** data type mapping keys to values. The central idea is to use a random-looking function to map keys to small numeric indices. The first published version of this is due to Dumey [Dum56].

Suppose we want to store n elements from a universe U of in a table with **keys** or **indices** drawn from an index space M of size m. Typically we

[1] An easy counting argument shows that almost all functions from U to M take $|U| \log |M|$ bits to represent, no matter how clever you are about choosing your representation. This forms the basis for **algorithmic information theory**, which *defines* an object as random if there is no way to reduce the number of bits used to express it.

assume $U = [|U|] = \{0 \dots |U| - 1\}$ and $M = [m] = \{0 \dots m - 1\}$.

If $|U| \leq m$, we can just use an array. Otherwise, we can map keys to positions in the array using a **hash function** $h : U \to M$. This necessarily produces **collisions**: pairs (x, y) with $h(x) = h(y)$, and any design of a hash table must include some mechanism for handling keys that hash to the same place. Typically this is a secondary data structure in each bin, but we may also place excess values in some other place. Typical choices for data structures are linked lists (**separate chaining** or just **chaining**) or secondary hash tables (see §7.3 below). Alternatively, we can push excess values into other positions in the same hash table according to some deterministic rule (**open addressing** or **probing**) or a second hash function (see §7.4).

For most of these techniques, the costs of insertions and searches will depend on how likely it is that we get collisions. An adversary that knows our hash function can always choose keys with the same hash value, but we can avoid that by choosing our hash function randomly.[2] Our ultimate goal is to do each search in $O(1 + n/m)$ expected time, which for $n \leq m$ will be much better than the $\Theta(\log n)$ time for pointer-based data structures like balanced trees or skip lists. The quantity n/m is called the **load factor** of the hash table and is often abbreviated as α.

All of this only works if we are working in a RAM (random-access machine model), where we can access arbitrary memory locations in time $O(1)$ and similarly compute arithmetic operations on $O(\log|U|)$-bit values in time $O(1)$. There is an argument that in reality any actual RAM machine requires either $\Omega(\log m)$ time to read one of m memory locations (routing costs) or, if one is particularly pedantic, $\Omega(m^{1/3})$ time (speed of light + finite volume for each location). We will ignore this argument.

We will try to be consistent in our use of variables to refer to the different parameters of a hash table. Table 7.1 summarizes the meaning of these variable names.

7.2 Universal hash families

A family of hash functions H is **2-universal** if for any $x \neq y$, $\Pr[h(x) = h(y)] \leq 1/m$ for a uniform random $h \in H$. It's **strongly 2-universal** if for any $x_1 \neq x_2 \in U$, $y_1, y_2 \in M$, $\Pr[h(x_1) = y_1 \wedge h(x_2) = y_2] = 1/m^2$ for a uniform

[2] In practice, hardly anybody every does this. Hash functions are instead chosen based on fashion and occasional experiments, often with additional goals like cryptographic security or speed. For cryptographic security, the **SHA** family is standard. For speed, **xxHash** (see https://cyan4973.github.io/xxHash/) is probably the fastest widely-used hash function with decent statistical properties.

U	Universe of all keys
$S \subseteq U$	Set of keys stored in the table
$n = \|S\|$	Number of keys stored in the table
M	Set of table positions
$m = \|M\|$	Number of table positions
$\alpha = n/m$	Load factor

Table 7.1: Hash table parameters

random $h \in H$. Another way to describe strong 2-universality is that the values of the hash function are uniformly distributed and pairwise-independent.

For $k > 2$, **k-universal** usually means **strongly k-universal**: Given distinct $x_1 \dots x_k$, and any $y_1 \dots y_k$, $\Pr[\forall i : h(x_i) = y_i] = m^{-k}$. This is equivalent to the $h(x_i)$ values for distinct x_i and randomly-chosen h having a uniform distribution and k-wise independence. It is possible to generalize the weak version of 2-universality to get a weak version of k-universality ($\Pr[h(x_i) \text{ are all equal}] \leq m^{-(k-1)}$), but this generalization is not as useful as strong k-universality.

To analyze universal hash families, it is helpful to have some notation for counting collisions. We'll mostly be doing counting rather than probabilities because it saves carrying around a lot of denominators. Since we are assuming uniform choices of h we can always get back probabilities by dividing by $|H|$. The particular notation we will be using follows §8.4.1 of [MR95]; the original paper of Carter and Wegman [CW77] uses essentially the same notation with slightly different formatting.

Let $\delta(x, y, h) = 1$ if $x \neq y$ and $h(x) = h(y)$, 0 otherwise. Abusing notation, we also define, for sets X, Y, and H, $\delta(X, Y, H) = \sum_{x \in X, y \in Y, h \in H} \delta(x, y, h)$; and allow lowercase variables to stand in for singleton sets, as in $\delta(x, Y, h) = \delta(\{x\}, Y, \{h\})$. Now the statement that H is 2-universal becomes $\forall x, y : \delta(x, y, H) \leq |H|/m$; this says that only $1/m$ of the functions in H cause any particular distinct x and y to collide.

If H includes all functions $U \to M$, we get equality: a random function gives $h(x) = h(y)$ with probability exactly $1/m$. But we might do better if each h tends to map distinct values to distinct places. The following lemma shows we can't do too much better:

Lemma 7.2.1. *For any family H, there exist x, y such that $\delta(x, y, H) \geq \frac{|H|}{m}\left(1 - \frac{m-1}{|U|-1}\right)$.*

Proof. We'll count collisions in the inverse image of each element z. Since

all distinct pairs of elements of $h^{-1}(z)$ collide with each other, we have

$$\delta(h^{-1}(z), h^{-1}(z), h) = \left|h^{-1}(z)\right| \cdot \left(\left|h^{-1}(z)\right| - 1\right).$$

Summing over all $z \in M$ gets all collisions, giving

$$\delta(U, U, h) = \sum_{z \in M} \left(\left|h^{-1}(z)\right| \cdot \left(\left|h^{-1}(z)\right| - 1\right)\right).$$

Use convexity or Lagrange multipliers to argue that the right-hand side is minimized subject to $\sum_z |h^{-1}(z)| = |U|$ when all pre-images are the same size $|U|/m$. It follows that

$$\begin{aligned}
\delta(U, U, h) &\geq \sum_{z \in M} \frac{|U|}{m} \left(\frac{|U|}{m} - 1\right) \\
&= m\frac{|U|}{m} \left(\frac{|U|}{m} - 1\right) \\
&= \frac{|U|}{m}(|U| - m).
\end{aligned}$$

If we now sum over all h, we get

$$\delta(U, U, H) \geq \frac{|H|}{m}|U|(|U| - m).$$

There are exactly $|U|(|U| - 1)$ ordered pairs x, y for which $\delta(x, y, H)$ might not be zero; so the Pigeonhole principle says some pair x, y has

$$\begin{aligned}
\delta(x, y, H) &\geq \frac{|H|}{m} \left(\frac{|U|(|U| - m)}{|U|(|U| - 1)}\right) \\
&= \frac{|H|}{m} \left(1 - \frac{m - 1}{|U| - 1}\right).
\end{aligned}$$

□

Since $1 - \frac{m-1}{|U|-1}$ is likely to be very close to 1, we are happy if we get the 2-universal upper bound of $|H|/m$.

Why we care about this: With a 2-universal hash family, chaining using linked lists costs $O(1 + n/m)$ expected time per operation. The reason is that the expected cost of an operation on some key x is proportional to the size of the linked list at $h(x)$ (plus $O(1)$ for the cost of hashing itself). But the expected size of this linked list is just the expected number of keys y in the dictionary that collide with x, which is exactly $\delta(x, S, H)/|H| \leq n/m$.

7.2.1 Linear congruential hashing

Universal hash families often look suspiciously like classic pseudorandom number generators. Here is a 2-universal hash family based on taking remainders. It is assumed that the universe U is a subset of $\mathbb{Z}_p$, the integers mod p; effectively, this just means that every element x of U satisfies $0 \leq x \leq p-1$.

Lemma 7.2.2. *Let $h_{ab}(x) = (ax + b \bmod p) \bmod m$, where $a \in \mathbb{Z}_p \setminus \{0\}, b \in \mathbb{Z}_p$, and p is a prime $\geq m$. Then $\{h_{ab}\}$ is 2-universal.*

Proof. Again, we count collisions. Split $h_{ab}(x)$ as $g(f_{ab}(x))$ where $f_{ab}(x) = ax + b \bmod p$ and $g(x) = x \bmod m$.

The intuition is that if we fix x and y and consider all $p(p-1)$ pairs a, b with $a \neq 0$, all $p(p-1)$ distinct pairs of values $r = f_{ab}(x)$ and $s = f_{ab}(y)$ are equally likely. We then show that feeding these values to g produces no more collisions than expected.

The formal statement of the intuition is that for any $0 \leq x, y \leq p-1$ with $x \neq y$, $\delta(x, y, H) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$.

To prove this, fix x and y, and consider some pair $r \neq s \in Z_p$. Then the equations $ax + b = r$ and $ay + b = s$ have a unique solution for a and b mod p (because $\mathbb{Z}_p$ is a finite field). Furthermore this solution has $a \neq 0$ since otherwise $f_{ab}(x) = f_{ab}(y) = b$. So the function $q(a, b) = \langle f_{ab}(x), f_{ab}(y) \rangle$ is a bijection between pairs a, b and pairs r, s. Any collisions will arise from applying g, giving $\delta(x, y, H) = \sum_{a,b} \delta(x, y, h_{ab}) = \sum_{r \neq s} \delta(r, s, g) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g)$.

Now we just need to count how many distinct r and s collide. There are p choices for r. For each r, the possible s that map to the same remainder mod m are in a set of the form $\{r', r' + m, r' + 2m, \ldots, r' + km\}$ where $r' = r \bmod m$ and $r' + km \leq p - 1$, which gives $k \leq (p-1)/m$. There are $k + 1$ elements of this set, but $s \neq r$ means we can only use k of them. This gives at most $k \leq (p-1)/m$ choices for s for each of the p choices for r. Multiplying out these bounds gives $\delta(x, y, H) = \delta(\mathbb{Z}_p, \mathbb{Z}_p, g) \leq p(p-1)/m$.

Since each choice of $a \neq 0$ and b occurs with probability $\frac{1}{p(p-1)}$, this gives a probability of collision of at most $1/m$. □

A difficulty with this hash family is that it requires doing modular arithmetic. A faster hash is given by Dietzfelbinger *et al.* [DHKP97], although it requires a slight weakening of the notion of 2-universality. For each k and ℓ they define a class $H_{k,\ell}$ of functions from $[2^k]$ to $[2^\ell]$ by defining

$$h_a(x) = (ax \bmod 2^k) \operatorname{div} 2^{k-\ell},$$

where $x \operatorname{div} y = \lfloor x/y \rfloor$. They prove [DHKP97, Lemma 2.4] that if a is a random *odd* integer with $0 < a < 2^\ell$, and $x \neq y$, $\Pr[h_a(x) = h_a(y)] \leq 2^{-\ell+1}$. This increases by a factor of 2 the likelihood of a collision, but any extra costs from this can often be justified in practice by the reduction in costs from working with powers of 2.

If we are willing to use more randomness (and more space), a method called **tabulation hashing** (§7.2.2) gives a simpler alternative that is 3-universal.

7.2.2 Tabulation hashing

Tabulation hashing [CW77] is a method for hashing fixed-length strings (or things that can be represented as fixed-length strings) into bit-vectors. The description here follows Patrascu and Thorup [PT12].

Let c be the length of each string in characters, and let s be the size of the alphabet. Initialize the hash function by constructing tables $T_1 \ldots T_c$ mapping characters to independent random bit-vectors of size $\lg m$. Define

$$h(x) = T_1[x_1] \oplus T_2[x_2] \oplus \ldots T_c[x_c],$$

where $\oplus$ represents bitwise exclusive OR (what ^ does in C-like languages).[3] This gives a family of hash functions that is 3-wise independent but not 4-wise independent.

Like many hash algorithms, tabulation hashing was already in use before it was formalized in general. **Zobrist hashing** [Zob70] is a special case of tabulation hashing used to represent positions in board games like Chess and Go, where $T_i[x_i]$ gives the contribution of having a piece x_i in position i. This is useful for games whose state changes slowly because of a homomorphic property of tabulation hashing: replacing x_i by x_i' while leaving all other x_j unchanged does not require recomputing the entire hash function, since $h[x'] = h[x] \oplus T_i[x_i] \oplus T_i'[x_i]$ in this case.

The intuition for why the hash values might be independent is that if we have a collection of strings, and each string brings in an element of T that doesn't appear in the other strings, then that element is independent of the hash values for the other strings and XORing it with the rest of the hash value gives a random bit string that is independent of the hash values of the other strings. In fact, we don't even need each string to include a unique

[3] Letting m be a power of 2 and using exclusive OR is convenient on real computers. If for some reason we don't like this approach, the same technique, with essentially the same analysis, works for arbitrary m if we replace bitwise XOR with addition mod m.

value; it's enough if we can order the strings so that each string gets a value that isn't represented among its predecessors.

More formally, suppose we can order the strings $x^1, x^2, \ldots, x^n$ that we are hashing so that each has a position i_j such that $x^j_{i_j} \neq x^{j'}_{i_j}$ for any $j' < j$, then we have, for each value v, $\Pr\left[h(x^j) = v \;\middle|\; h(x^{j'}) = v_{j'}, \forall j' < j\right] = 1/m$. It follows that the hash values are independent:

$$\begin{aligned}\Pr\left[h(x^1) = v_1, h(x^2) = v_2, \ldots, h(x^n) = v_n)\right] &= \prod_{j=1}^{n} \Pr\left[h(x^j) = v_j \;\middle|\; h(x^1) = v_1 \ldots h(x^{j-1}) = v_{j-1}\right] \\ &= \frac{1}{m^n} \\ &= \prod_{j=1}^{n} \Pr\left[h(x^j) = v_j\right].\end{aligned}$$

Now we want to show that when $n = 3$, this actually works for all possible distinct strings x, y, and z. Let S be the set of indices i such that $y_i \neq x_i$, and similarly let T be the set of indices i such that $z_i \neq x_i$; note that both sets must be non-empty, since $y \neq x$ and $z \neq x$. If $S \setminus T$ is nonempty, then (a) there is some index i in T where $z_i \neq x_i$, and (b) there is some index j in $S \setminus T$ where $y_i \neq x_i = z_i$; in this case, ordering the strings as x, z, y gives the independence property above. If $T \setminus S$ is nonempty, order them as x, y, z instead. Alternatively, if $S = T$, then $y_i \neq z_i$ for some i in S (otherwise $y = z$, since they both equal x on all positions outside S). In this case, x_i, y_i, and z_i are all distinct.

For $n = 4$, we can have strings `aa`, `ab`, `ba`, and `bb`. If we take the bitwise exclusive OR of all four hash values, we get zero, because each character is included exactly twice in each position. So the hash values are not independent, and we do not get 4-independence in general.

However, even though the outputs of tabulation hashing are not 4-independent, most reasonably small sets of inputs do give independence. This can be used to show various miraculous properties like working well for the cuckoo hashing algorithm described in §7.4.

7.3 FKS hashing

The FKS hash table, named for Fredman, Komlós, and Szemerédi [FKS84], is a method for storing a static set S so that we never pay more than constant time for search (not just in expectation), while at the same time

not consuming too much space. The assumption that S is static is critical, because FKS chooses hash functions based on the elements of S.

If we were lucky in our choice of S, we might be able to do this with standard hashing. A **perfect hash function** for a set $S \subseteq U$ is a hash function $h : U \to M$ that is injective on S (that is, $x \neq y \to h(x) \neq h(y)$ when $x, y \in S$). Unfortunately, we can only count on finding a perfect hash function if m is large:

Lemma 7.3.1. *If H is 2-universal and $|S| = n$ with $\binom{n}{2} \leq m$, then there is a perfect $h \in H$ for S.*

Proof. Let h be chosen uniformly at random from H. For each unordered pair $x \neq y$ in S, let X_{xy} be the indicator variable for the even that $h(x) = h(y)$, and let $C = \sum_{x \neq y} X_{xy}$ be the total number of collisions in S. Each X_{xy} has expectation at most $1/m$, so $\mathrm{E}[C] \leq \binom{n}{2}/m < 1$. But we we can write $\mathrm{E}[C]$ as $\mathrm{E}[C \mid C = 0] \Pr[C = 0] + \mathrm{E}[C \mid C \geq 1] \Pr[C \neq 0] \geq \Pr[C \neq 0]$. So $\Pr[C \neq 0] \leq \binom{n}{2}/m < 1$, giving $\Pr[C = 0] > 0$. But if C is zero with nonzero probability, there must be some h that makes it 0. That h is perfect for S. □

Essentially the same argument shows that if $\alpha\binom{n}{2} \leq m$, then $\Pr[h \text{ is perfect for } S] \geq 1 - \alpha$. This can be handy if we want to find a perfect hash function and not just demonstrate that it exists.

Using a perfect hash function, we get $O(1)$ search time using $O(n^2)$ space. But we can do better by using perfect hash functions only at the second level of our data structure, which at top level will just be an ordinary hash table. This is the idea behind the Fredman-Komlós-Szemerédi (FKS) hash table [FKS84].

The short version is that we hash to $n = |S|$ bins, then rehash perfectly within each bin. The top-level hash table stores a pointer to a header for each bin, which gives the size of the bin and the hash function used within it. The i-th bin, containing n_i elements, uses $O(n_i^2)$ space to allow perfect hashing. The total size is $O(n)$ as long as we can show that $\sum_{i=1}^{n} n_i^2 = O(n)$. The time to do a search is $O(1)$ in the worst case: $O(1)$ for the outer hash plus $O(1)$ for the inner hash.

Theorem 7.3.2. *The FKS hash table uses $O(n)$ space.*

Proof. Suppose we choose $h \in H$ as the outer hash function, where H is

some 2-universal family of hash functions. Compute:

$$\begin{aligned}\sum_{i=1}^{n} n_i^2 &= \sum_{i=1}^{n}(n_i + n_i(n_i - 1)) \\ &= n + \delta(S, S, h).\end{aligned}$$

The last equality holds because each ordered pair of distinct values in S that map to the same bucket i corresponds to exactly one collision in $\delta(S, S, h)$.

Since H is 2-universal, we have $\delta(S, S, H) \leq |H|\frac{|S|(|S|-1)}{n} = |H|\frac{n(n-1)}{n} = |H|(n-1)$. But then the Pigeonhole principle says there exists some $h \in H$ with $\delta(S, S, h) \leq \frac{1}{|H|}\delta(S, S, H) = n - 1$. Choosing this h gives $\sum_{i=1}^{n} n_i^2 \leq n + (n-1) = 2n - 1 = O(n)$. □

If we want to find a good h quickly, increasing the size of the outer table to n/α gives us a probability of at least $1 - \alpha$ of getting a good one, using essentially the same argument as for perfect hash functions. More generally, it's possible to adapt FKS hashing to work with dynamic data sets by growing the main table and each subtable as needed to keep the 2-lookup property while maintaining $O(1)$ expected amortized cost per insertion. This is know as **dynamic perfect hashing** and was studied by Dietzfelbinger *et al.* [DKM+94]. In practice the overhead of this approach is higher than other 2-lookup schemes, such as the one described in the next section.

7.4 Cuckoo hashing

Cuckoo hashing [PR04] is a hash table implementation that uses a single table and guarantees at most 2 probes per lookup, with $O(1)$ expected amortized cost per insertion.

The name comes from the **cuckoo**, a bird notorious for stealing space for their own eggs in other birds' nests. In cuckoo hashing, newly-inserted elements may steal slots from other elements, forcing those elements to find an alternate nest.

The formal mechanism is to use two hash functions h_1 and h_2, and store each element x in one of the two positions $h_1(x)$ or $h_2(x)$. This may require moving other elements to their alternate locations to make room. But the payoff is that each search takes only two reads, which can even be done in parallel. This is optimal by a lower bound of Pagh [Pag01], which also shows a matching upper bound for static dictionaries using a different technique.

Cuckoo hashing was invented by Pagh and Rodler [PR04]. The version described here is based on a simplified version from notes of Pagh [Pag06].

The main difference is that it uses just one table instead of the two tables—one for each hash function—in [PR04].

7.4.1 Structure

We have a table T of size n, with two separate, independent hash functions h_1 and h_2. These functions are assumed to be k-universal for some sufficiently large value k; as long as we never look at more than k values at once, this means we can treat them effectively as random functions. In practice, using crummy hash functions seems to work just fine, a common property of hash tables. There are also specific hash functions that have been shown to work with particular variants of cuckoo hashing [PR04, PT12]. We will avoid these issues by assuming that our hash functions are actually random.

Every key x is stored either in $T[h_1(x)]$ or $T[h_2(x)]$. So the search procedure just looks at both of these locations and returns whichever one contains x (or fails if neither contains x).

To insert a value $x_1 = x$, we must put it in $T[h_1(x_1)]$ or $T[h_2(x_1)]$. If one or both of these locations is empty, we put it there. Otherwise we have to kick out some value that is in the way (this is the "cuckoo" part of cuckoo hashing, named after the bird that leaves its eggs in other birds' nests). We do this by letting $x_2 = T[h_1(x_1)]$ and writing x_1 to $T[h_1(x_1)]$. We now have a new "nestless" value x_2, which we swap with whatever is in $T[h_2(x_2)]$. If that location was empty, we are done; otherwise, we get a new value x_3 that we have to put in $T[h_1(x_3)]$ and so on. The procedure terminates when we find an empty spot or if enough iterations have passed that we don't expect to find an empty spot, in which case we rehash the entire table. This process can be implemented succinctly as shown in Algorithm 7.1.

A detail not included in the above code is that we always rehash (in theory) after m^2 insertions; this avoids potential problems with the hash functions used in the paper not being universal enough. We will avoid this issue by assuming that our hash functions are actually random (instead of being approximately n-universal with reasonably high probability). For a more principled analysis of where the hash functions come from, see [PR04]. An alternative hash family that is known to work for a slightly different variant of cuckoo hashing is tabulation hashing, as described in §7.2.2; the proof that this works is found in [PT12].

```
1  procedure insert(x)
2      if T(h_1(x)) = x or T(h_2(x)) = x then
3          return
4      pos ← h_1(x)
5      for i ← 1...n do
6          if T[pos] = ⊥ then
7              T[pos] ← x
8              return
9          x ⇄ T[pos]
10         if pos = h_1(x) then
11             pos ← h_2(x)
12         else
13             pos ← h_1(x)
14     If we got here, rehash the table and reinsert x.
```

Algorithm 7.1: Insertion procedure for cuckoo hashing. Adapted from [Pag06]

7.4.2 Analysis

The main question is how long it takes the insertion procedure to terminate, assuming the table is not too full.

First let's look at what happens during an insert if we have many nestless values. We have a sequence of values $x_1, x_2, \ldots$, where each pair of values x_i, x_{i+1} collides in h_1 or h_2. Assuming we don't reach the loop limit, there are three main possibilities (the leaves of the tree of cases below):

1. Eventually we reach an empty position without seeing the same key twice.

2. Eventually we see the same key twice; there is some i and $j > i$ such that $x_j = x_i$. Since x_i was already moved once, when we reach it the second time we will try to move it back, displacing x_{i-1}. This process continues until we have restored x_2 to $T[h_1(x_1)]$, displacing x_1 to $T[h_2(x_1)]$ and possibly creating a new sequence of nestless values. Two outcomes are now possible:

   (a) Some x_ℓ is moved to an empty location. We win!

   (b) Some x_ℓ is moved to a location we've already looked at. We lose!

> We find we are playing musical chairs with more players than chairs, and have to rehash.

Let's look at the probability that we get the last case, a **closed loop**. Following the argument of Pagh and Rodler, we let v be the number of distinct nestless keys in the loop. Since v includes x_1 and at least one other element blocking x_1 from being inserted at $T[h_1(x_1)]$, v is at least 2. We can now count how many different ways such a loop can form, and argue that in each case we include enough information to reconstruct $h_1(u_i)$ and $h_2(u_i)$ for each of a specific set of unique elements $u_1, \ldots u_v$.

Formally, this means that we are expression the closed-loop case as a union of many specific closed loops, and then bounding the probability of each of these specific closed-loop events by the probability of the event that h_1 and h_2 select the right values to make this particular closed loop possible. Then we apply the union bound.

To describe each of the specific events, we'll provide this information:

- The v elements $u_1, \ldots u_v$. Since we can fix $u_1 = x_1$, this leaves $v - 1$ choices from S, giving $n^{(v-1)}$ possibilities (we are overcounting by allowing duplicates, but that's not a problem for an upper bound). We'll require that the other u_i for $i > 1$ appear in the list in the same order they first appear in the sequence $x_1, x_2, \ldots$.

- The $v - 1$ locations we are trying to fit these v elements into. There are at most $m^{(v-1)}$ choices for these. Again we order these locations by order of first appearance.

- The values of i, j, and ℓ. These allow us to identify which segments of the sequence $x_1, x_2, \ldots$ correspond to new values u_i and which are old values repeated (possibly in reverse order).

  There are at most v choices for i and j (because we are still in the initial segment with no repeats), and at most $2v$ choices for ℓ if we count carefully (because we either land on either the initial no-duplicate sequence starting with x_1 or the second no-duplicate sequence starting with the second occurrence of x_1).

  All together, these give $2v^3$ choices.

- For each $i \neq 1$, whether the first occurrence of u_i appears in $h_1(u_i)$ or $h_2(u_i)$. This gives 2^{v-1} choices, and allows us to correctly identify

$h_1(u_i)$ or $h_2(u_1)$ from the value of u_i and its first location and the other hash value for u_i given the next location in the list.[4]

Multiplying everything out gives at most $2v^3(2nm)^{(v-1)}$ choices of closed loops with v unique elements. Since each particular loop allows us to determine both h_1 and h_2 for all v of its elements, the probability that we get exactly these hash values (so that the loop occurs) is m^{-2v}. Summing over all closed loops with v elements gives a total probability of

$$\begin{aligned} 2v^3(2nm)^{v-1} \cdot m^{-2v} &= 2v^3(2n)^{v-1}m^{-v-1} \\ &= 2v^3(2n/m)^{v-1}m^{-2}. \end{aligned}$$

Now sum over all $v \geq 2$. We get

$$m^{-2}\sum_{v=2}^{n} 2v^3(2n/m)^{v-1} < m^{-2}\sum_{v=2}^{\infty} 2v^3(2n/m)^{v-1}.$$

The series converges if $2n/m < 1$, so for any fixed $\alpha < 1/2$, the probability of any closed loop forming is $O(m^{-2})$.

If we do hit a closed loop, then we pay $O(m)$ time to scan the existing table and create a new empty table, and $O(n) = O(m)$ time on average to reinsert all the elements into the new table, assuming that this reinsertion process doesn't generate any more closed loops and that the average cost of an insertion that doesn't produce a closed loop is $O(1)$, which we will show below. But the rehashing step only fails with probability $O(nm^{-2}) = O(m^{-1})$, so if it does fail we can just try again until it works, and the expected total cost is still $O(m)$. Since we pay this $O(m)$ for each insertion with probability $O(m^{-2})$, this adds only $O(m^{-1})$ to the expected cost of a single insertion.

Now we look at what happens if we don't get a closed loop. This doesn't force us to rehash, but if the path is long enough, we may still pay a lot to do an insertion.

It's a little messy to analyze the behavior of keys that appear more than once in the sequence, so the trick used in the paper is to observe that for any sequence of nestless keys $x_1 \ldots x_p$, there is a subsequence of size $p/3$ with no repetitions that starts with x_1. This will be either the sequence S_1 given by $x_1 \ldots x_{j-1}$—the sequence starting with the first place we try to insert x_1–or S_2 given by $x_1 = x_{i+j-1} \ldots x_p$, the sequence starting with the second place we try to insert x_1. Between these we have a third sequence R where we

[4]The original analysis in [PR04] avoids this by alternating between two tables, so that we can determine which of h_1 or h_2 is used at each step by parity.

revert some of the moves made in S_1. Because $|S_1| + |R| + |S_2| \geq p$, at least one of these three subsequences has size $p/3$. But $|R| \leq |S_1|$, so it must be either S_1 or S_2.

We can then argue that the probability that we get a sequence of v distinct keys in either S_1 or S_2 is at most $2(n/m)^{v-1}$. The $(n/m)^{v-1}$ is because we need to hit a nonempty spot (which happens with probability at most n/m) for the first $v-1$ elements in the path, and since we assume that our hash functions are random, the choices of these $v-1$ spots are all independent. The 2 is from the union bound over S_1 and S_2. If T is length of the longer of S_1 or S_2, we get $\mathrm{E}[T] = \sum_{v=1}^{\infty} \Pr[T \geq v] \leq \sum_{v=1}^{\infty} 2(n/m)^{v-1} = O(1)$, assuming n/m is bounded by a constant less than 1. Since we already need $n/m \leq 1/2$ to avoid the bad closed-loop case, we can use this here as well. We have to multiply $\mathrm{E}[T]$ by 3 to get the bound on the actual path, but this disappears into $O(1)$.

An annoyance with cuckoo hashing is that it has high space overhead compared to more traditional hash tables: in order for the first part of the analysis above to work, the table must be at least half empty. This can be avoided at the cost of increasing the time complexity by choosing between d locations instead of 2. This technique, due to Fotakis *et al.* [FPSS03], is known as **d-ary cuckoo hashing**. For a suitable choice of d, it uses $(1+\epsilon)n$ space and guarantees that a lookup takes $O(1/\epsilon)$ probes, while insertion takes $(1/\epsilon)^{O(\log\log(1/\epsilon))}$ steps in theory and appears to take $O(1/\epsilon)$ steps in practice according to experiments done by the authors.

7.5 Practical issues

Most hash functions used in practice do not have very good theoretical guarantees, and indeed we have assumed in several places in this chapter that we are using genuinely random hash functions when we would expect our actual hash functions to be at most 2-universal. There is some justification for doing this if there is enough entropy in the set of keys S. A proof of this for many common applications of hash functions is given by Chung *et al.* [CMV13].

Even taking into account these results, hash tables that depend on strong properties of the hash function may behave badly if the user supplies a crummy hash function. For this reason, many library implementations of hash tables are written defensively, using algorithms that respond better in bad cases. See `https://svn.python.org/projects/python/trunk/Objects/dictobject.c` for an example of a widely-used hash table implementation

chosen specifically because of its poor theoretical characteristics.

For large hash tables, local probing schemes are often faster than chaining or cuckoo hashing, because it is likely that all of the locations probed to find a particular value will be on the same virtual memory page. This means that a search for a new value usually requires one cache miss instead of two. **Hopscotch hashing** [HST08] combines ideas from linear probing and cuckoo hashing to get better performance than both in practice.

7.6 Bloom filters

See [MU17, §5.5.3] for basics and a formal analysis or http://en.wikipedia.org/wiki/Bloom_filter for many variations and the collective wisdom of the unwashed masses. The presentation here mostly follows [MU17].

7.6.1 Construction

Bloom filters are a highly space-efficient randomized data structure invented by Burton H. Bloom [Blo70] for storing sets of keys, with a small probability for each key not in the set that it will be erroneously reported as being in the set.

Suppose we have k independent hash functions $h_1, h_2, \ldots, h_k$. Our memory store A is a vector of m bits, all initially zero. To store a key x, set $A[h_i(x)] = 1$ for all i. To test membership for x, see if $A[h_i(x)] = 1$ for all i. The membership test always gives the right answer if x is in fact in the Bloom filter. If not, we might decide that x is in the Bloom filter anyway, just because we got lucky.

7.6.2 False positives

The probability of such **false positives** can be computed in two steps: first, we estimate how many of the bits in the Bloom filter are set after inserting n values, and then we use this estimate to compute a probability that any fixed x shows up when it shouldn't.

If the h_i are close to being independent random functions,[5] then with n entries in the filter we have $\Pr[A[i] = 1] = 1 - (1 - 1/m)^{kn}$, since each of

[5] We are going to sidestep the rather deep swamp of how plausible this assumption is and what assumption we should be making instead. However, it is known [KM08] that starting with two sufficiently random-looking hash functions h and h' and setting $h_i(x) = h(x) + ih'(x)$ works.

the kn bits that we set while inserting the n values has one chance in m of hitting position i.

We'd like to simplify this using the inequality $1 + x \leq e^x$, but it goes in the wrong direction; instead, we'll use $1 - x \geq e^{-x-x^2}$, which holds for $0 \leq x \leq 0.683803$ and in our application holds for $m \geq 2$. This gives

$$\begin{aligned}\Pr[A[i] = 1] &\leq 1 - (1 - 1/m)^{kn} \\ &\leq 1 - e^{-kn(1/m)(1+1/m)} \\ &= 1 - e^{-k\alpha(1+1/m)} \\ &= 1 - e^{-k\alpha'}\end{aligned}$$

where $\alpha = n/m$ is the load factor and $\alpha' = \alpha(1 + 1/m)$ is the load factor fudged upward by a factor of $1 + 1/m$ to make the inequality work.

Suppose now that we check to see if some value x that we never inserted in the Bloom filter appears to be present anyway. This occurs if $A[h_i(x)] = 1$ for all i. Since each $A[h_i(x)]$ is an independent sample from A, the probability that they all come up 1 conditioned on A is

$$\left(\frac{\sum A[i]}{m}\right)^k. \tag{7.6.1}$$

We have an upper bound $\mathrm{E}\left[\sum A[i]\right] \leq m\left(1 - e^{-k\alpha'}\right)$, and if we were born luckier, we might be able to get an upper bound on the expectation of (7.6.1) by applying Jensen's inequality to the function $f(x) = x^k$. But sadly this inequality also goes in the wrong direction, because f is convex for $k > 1$. So instead we will prove a concentration bound on $S = \sum A[i]$.

Because the $A[i]$ are not independent, we can't use off-the-shelf Chernoff bounds. Instead, we rely on McDiarmid's inequality. Our assumption is that the locations of the kn ones that get written to A are independent. Furthermore, changing the location of one of these writes changes S by at most 1. So McDiarmid's inequality (5.3.13) gives $\Pr[S \geq \mathrm{E}[S] + t] \leq e^{-2t^2/kn}$, which is bounded by n^{-c} for $t \geq \sqrt{\frac{1}{2}ckn \log n}$. So as long as a reasonably large fraction of the array is likely to be full, the relative error from assuming $S = \mathrm{E}[S]$ is likely to be small. Alternatively, if the array is mostly empty, then we don't care about the relative error so much because the probability of getting a false positive will already be exponentially small as a function of k.

So let's assume for simplicity that our false positive probability is exactly $(1 - e^{-k\alpha'})^k$. We can choose k to minimize this quantity for fixed α' by

doing the usual trick of taking a derivative and setting it to zero; to avoid weirdness with the k in the exponent, it helps to take the logarithm first (which doesn't affect the location of the minimum), and it further helps to take the derivative with respect to $x = e^{-\alpha' k}$ instead of k itself. Note that when we do this, $k = -\frac{1}{\alpha'} \ln x$ still depends on x, and we will deal with this by applying this substitution at an appropriate point.

Compute

$$\begin{aligned}\frac{d}{dx} \ln\left((1-x)^k\right) &= \frac{d}{dx} k \ln(1-x) \\ &= \frac{d}{dx}\left(-\frac{1}{\alpha'} \ln x\right) \ln(1-x) \\ &= -\frac{1}{\alpha'}\left(\frac{\ln(1-x)}{x} - \frac{\ln x}{1-x}\right).\end{aligned}$$

Setting this to zero gives $(1-x)\ln(1-x) = x \ln x$, which by symmetry has the unique solution $x = 1/2$, giving $k = \frac{1}{\alpha'} \ln 2$.

In other words, to minimize the false positive rate for a known load factor α, we want to choose $k = \frac{1}{\alpha'} \ln 2 = \frac{1}{\alpha(1+1/m)} \ln 2$, which makes each bit one with probability approximately $1 - e^{-\ln 2} = \frac{1}{2}$. This makes intuitive sense, since having each bit be one or zero with equal probability maximizes the entropy of the data.

The probability of a false positive for a given key is then $2^{-k} = 2^{-\ln 2/\alpha'}$. For a given maximum false positive rate ϵ, and assuming optimal choice of k, we need to keep $\alpha' \leq \frac{\ln^2 2}{\ln(1/\epsilon)}$ or $\alpha \leq \frac{\ln^2 2}{(1+1/m)\ln(1/\epsilon)}$.

Another way to look at this is that if we fix ϵ and n, we need $m/(1 + 1/m) \geq n \cdot \frac{\ln(1/\epsilon)}{\ln^2 2} \approx 1.442695 \cdot n \lg(1/\epsilon)$, which works out to $m \geq 1.442695 \cdot n \lg(1/\epsilon) + O(1)$. This is very good for constant ϵ.

Note that for this choice of m, we have $\alpha = O(1/\ln(1/\epsilon))$, giving $k = O(\log(1/\epsilon))$. So for polynomial ϵ, we get $k = O(\log n)$. This is closer to the complexity of tree lookups than hash table lookups, so the main payoff for a sequential implementation is that we don't have to store full keys.

7.6.3 Comparison to optimal space

If we wanted to design a Bloom-filter-like data structure from scratch and had no constraints on processing power, we'd be looking for something that stored an index of size $\lg M$ into a family of subsets $S_1, S_2, \ldots S_M$ of our universe of keys U, where $|S_i| \leq \epsilon |U|$ for each i (giving the upper bound on

the false positive rate)[6] and for any set $A \subseteq U$ of size n, $A \subseteq S_i$ for at least one S_i (allowing us to store A).

Let $N = |U|$. Then each set S_i covers $\binom{\epsilon N}{n}$ of the $\binom{N}{n}$ subsets of size n. If we could get them to overlap optimally (we can't), we'd still need a minimum of $\binom{N}{n} \Big/ \binom{\epsilon N}{n} = (N)_n/(\epsilon N)_n \approx (1/\epsilon)^n$ sets to cover everybody, where the approximation assumes $N \gg n$. Taking the log gives $\lg M \approx n \lg(1/\epsilon)$, meaning we need about $\lg(1/\epsilon)$ bits per key for the data structure. Bloom filters use $1/\ln 2$ times this.

There are known data structures that approach this bound asymptotically. The first of these, due to Pagh *et al.* [PPR05] also has other desirable properties, like supporting deletions and faster lookups if we can't look up bits in parallel.

More recently, Fan *et al.* [FAKM14] have described a variant of cuckoo hashing (see §7.4) called a **cuckoo filter**. This is a cuckoo hash table that, instead of storing full keys x, stores **fingerprints** $f(x)$, where f is a hash function with ℓ-bit outputs. False positives now arise if we happen to hash a value x' with $f(x') = f(x)$ to the same location as x. If f is drawn from a 2-universal family, this occurs with probability at most $2^{-\ell}$. So the idea is that by accepting an ϵ small rate of false positives, we can shrink the space needed to store each key from the full key length to $\lg(1/\epsilon) = \ln(1/\epsilon)/\ln 2$, the asymptotic minimum.

One complication is that, since we are throwing away the original key x, when we displace a key from $h_1(x)$ to $h_2(x)$ or vice versa, we can't recompute $h_1(x)$ and $h_2(x)$ for arbitrary h_1 and h_2. The solution proposed by Fan *et al.* is to let $h_2(x) = h_1(x) \oplus g(f(x))$, where g is a hash function that depends only on the fingerprint. This means that when looking at a fingerprint $f(x)$ stored in position i, we don't need to know whether i is $h_1(x)$ or $h_2(x)$, since whichever it is, the other location will be $i \oplus g(f(x))$. Unfortunately, this technique and some other techniques used in the paper to crunch out excess empty space break the standard analysis of cuckoo hashing, so the authors can only point to experimental evidence that their data structure actually

[6] Technically, this gives a weaker bound on false positives. For standard Bloom filters, assuming random hash functions, each key individually has at most an ϵ probability of appearing as a false positive. The hypothetical data structure we are considering here—which is effectively deterministic—allows the set of false positives to depend directly on the set of keys actually inserted in the data structure, meaning that the adversary could arrange for a specific key to appear as a false positive with probability 1 by choosing appropriate keys to insert. So this argument may underestimate the space needed to get make the false positives less predictable. On the other hand, we aren't charging the Bloom filter for the space needed to store the hash functions, which could be quite a bit if they are genuine random functions.

works. However, a variant of this data structure has been shown to work by Eppstein [Epp16].

7.6.4 Applications

Historically, Bloom filters were invented to act as a way of filtering queries to a database table through fast but expensive[7] RAM before looking up the actual values on a slow but cheap tape drive. Nowadays the cost of RAM is low enough that this is less of an issue in most cases, but Bloom filters are still popular in networking and in distributed databases.

In networking, Bloom filters are useful in building network switches, where incoming packets need to be matched against routing tables in fractions of a nanosecond. Bloom filters work particularly well for this when implemented in hardware, since the k hash functions can be computed in parallel. False positives, if infrequent enough, can be handled by some slower backup mechanism.

In distributed databases, Bloom filters are used in the **Bloomjoin** algorithm [ML86]. Here we want to do a join on two tables stored on different machines (a join is an operation where we find all pairs of rows, one in each table, that match on some common key). A straightforward but expensive way to do this is to send the list of keys from the smaller table across the network, then match them against the corresponding keys from the larger table. If there are n_s rows in the smaller table, n_b rows in the larger table, and j matching rows in the larger table, this requires sending n_s keys plus j rows. If instead we send a Bloom filter representing the set of keys in the smaller table, we only need to send $n \lg(1/\epsilon)/\ln 2$ bits for the Bloom filter plus an extra ϵn_b rows on average for the false positives. This can be cheaper than sending full keys across if the number of false positives is reasonably small.

7.6.5 Counting Bloom filters

It's not hard to modify a Bloom filter to support deletion. The basic trick is to replace each bit with a counter, so that whenever a value x is inserted, we increment $A[h_i(x)]$ for all i and when it is deleted, we decrement the same locations. The search procedure now returns $\min_i A[h_i(x)]$ (which means that it principle it can even report back multiplicities, though with some probability of reporting a value that is too high). To avoid too much space overhead, each array location is capped at some small maximum value c;

[7] As much as \$0.10/bit in 1970.

once it reaches this value, further increments have no effect. The resulting structure is called a **counting Bloom filter**, due to Fan *et al.* [FCAB00].

We can only expect this to work if our chance of hitting the cap is small. Fan *et al.* observe that the probability that the m table entries include one that is at least c after n insertions is bounded by

$$\begin{aligned} m\binom{nk}{c}\frac{1}{m^c} &\le m\left(\frac{enk}{c}\right)^c\frac{1}{m^c} \\ &= m\left(\frac{enk}{cm}\right)^c \\ &= m(ek\alpha/c)^c. \end{aligned}$$

(This uses the bound $\binom{n}{k} \le \left(\frac{en}{k}\right)^k$, which follows from Stirling's formula.)

For $k = \frac{1}{\alpha}\ln 2$, this is $m(e\ln 2/c)^c$. For the specific value of $c = 16$ (corresponding to 4 bits per entry), they compute a bound of $1.37 \times 10^{-15}m$, which they argue is minuscule for all reasonable values of m (it's a systems paper).

The possibility that a long chain of alternating insertions and deletions might produce a false negative due to overflow is considered in the paper, but the authors state that "the probability of such a chain of events is so low that it is much more likely that the proxy server would be rebooted in the meantime and the entire structure reconstructed." An alternative way of dealing with this problem is to never decrement a maxed-out register. This never produces a false negative, but may cause the filter to slowly fill up with maxed-out registers, producing a higher false-positive rate.

A fancier variant of this idea is the **spectral Bloom filter** of Cohen and Matias [CM03], which uses larger counters to track multiplicities of items. The essential idea here is that we can guess that the number of times a particular value x was inserted is equal to $\min_{i=1}^{k} A[h_i(x)]$, with some extra tinkering to detect errors based on deviations from the typical joint distribution of the $A[h_i(x)]$ values. An even more sophisticated approach gives the count-min sketches of the next section.

7.7 Data stream computation

In the **data stream** model, we are given a huge flood of data—far too big to store—in a single pass, and want to incrementally build a small data structure, called a **sketch**, that will allow us to answer statistical questions about the data after we've processed it all. The motivation is the existence

of data sets that are too large to store at all (network traffic statistics), or too large to store in fast memory (very large database tables). By building an appropriate small data structure using a single pass through the data, we can still answer queries about with some loss of accuracy. Examples we will consider include estimating the size of a set presented over time with possible duplicate elements (§7.7.1) or more general statistical queries based on aggregate counts of some sort (§7.7.2).

In each of these cases, the answers we get will be approximate. We will measure the quality of the approximation in terms of parameters (δ, ϵ), where we demand a relative error of at most ϵ with probability at least $1 - \delta$. We'd also like our data structure to have size at most polylogarithmic in the number of samples n and polynomial in δ and ϵ.

7.7.1 Cardinality estimation

The **cardinality estimation** or **count-duplicates** problem involves seeing a sequence of values $x_1, x_2, \ldots, x_n$ and asking to compute the number of *unique* values in this sequence.

Without the uniqueness constraint, this is trivial: just keep a counter. With the uniqueness constraint, exact counting is much harder, since any data structure that lets us detect if we see a new element also lets us test for membership. But if we are willing to accept an approximation, we can get around this by using hashing and then tracking statistical properties of the hashed values. To keep the analysis simple, we will assume that our hashing function is a random function, and not charge for storing its parameters.

Many algorithms of this type are based on a tool called a **Flajolet-Martin sketch** [FNM85]. The simplest version of this is that we pick a random hash function h, and use it to generate a geometric random variable R_i for each x_i by counting the number of trailing zeroes in the binary representation of $h(x_i)$. We then track $R = \max_i R_i$ and estimate the number n of unique x_i using 2^R.

The intuition for why this counts unique x_i is that sending in the same x_i twice produces the same $R_i = h(x_i)$ both times, so the maximum R is unaffected.

It's easy to show that this gives a reasonably good approximation to n. For the upper bound, given n samples, the expected number of samples with $R_i \geq k$ is $n \cdot 2^{-k}$, so for $k = \lg n + \ell$, the probability that $2^R \geq 2^k$ is at most $2^{-\ell}$ by Markov's inequality. For the lower bound, we have $\Pr[R_i < k] = 1 - 2^{-k}$ so $\Pr[\forall i : R_i < k] = (1 - 2^{-k})^n \leq e^{-n2^{-k}}$, which gives $\Pr[\max R_i < \lg n - \ell] \leq e^{-2^\ell}$. The lower bound is a bit stronger than the

upper bound, but in both directions we get at least a constant probability of being within a (large) constant factor of the correct answer. The expected value can also be shown to converge to the correct value for large enough n, after multiplying by a small correction factor ϕ that compensates for systematic round-off error caused by quantizing to powers of 2.

To improve the constants, Flajolet and Martin proposed a technique they called **Probabilistic Counting with Stochastic Averaging** or **PCSA**. This splits the incoming stream into m buckets using a second hash function (or, in practice, the leading bits of the first one). Each bucket gives its own estimate $\hat{n}_i = m\phi 2^R$, then these estimates are averaged to produce to final estimate. Flajolet and Martin show that, with an appropriate multiplier, this estimate has a typical error of $O(1/\sqrt{m})$. This analysis is pretty involved so we will not repeat it here.

The original PCSA algorithm is not used much in practice. More popular is a descendant called **HyperLogLog** [FFGM07] that replaces the arithmetic mean $\frac{1}{m}\sum_i \hat{n}_i$ with a harmonic mean

$$\frac{1}{\frac{1}{m}\sum_i \frac{1}{\hat{n}_i}}.$$

As with PCSA, HyperLogLog requires using some carefully calculated corrections to get an unbiased estimator. This can be avoided using an auxiliary counter that is updated whenever the main data structure changes, which also gives some improvement in the accuracy of the estimate. This mechanism was originally proposed independently by Cohen [Coh15] and Ting [Tin14], although the description we give here is largely drawn from a more recent paper by Pettie *et al.* [PWY20], which refers to this approach as the **martingale transformation** of the original data structure.

What this transformation does is observe that for each state s of the HyperLogLog (or whatever) data structure, there is a probability p_s that the next unique element will send it to a new state s'. If we can calculate this probability, then we can update our estimated count $\hat{\lambda}$ by increasing it by $1/p_s$ when the change occurs. Because each new unique element gives an expected increase of exactly $p_s \cdot \frac{1}{p_s} = 1$, this makes the expected value in the counter exactly equal to the actual count. (The connection between this and martingales is that we just showed that $\hat{\lambda}_t - \lambda_t$ is a martingale where $\hat{\lambda}_t$ is the value of the counter after t steps and λ_t is the actual number of unique elements.)

The only tricky part here is computing p_s. For HyperLogLog, there is a $1/m$ chance that our new unique element lands in each of the m buckets, and

it lands in a bucket that currently stores r_i, the probability that we increase r_i is exactly 2^{-r_i-1}. This immediately gives

$$p_s = \frac{1}{m}\sum_i 2^{-r_i-1}$$

and thus

$$\frac{1}{p_s} = \frac{1}{\frac{1}{m}\sum_i 2^{-r_i-1}},$$

which looks suspiciously like the harmonic mean used on the final estimates in HyperLogLog. As with the original HyperLogLog, it is possible to show that the typical relative error for this sketch is $O(1/\sqrt{m})$. See [PWY20] for more details and some further improvements.

If we don't care about practical engineering issues, there is a known asymptotically optimal solution to the cardinality estimation problem [KNW10], which doesn't even require assuming a random oracle, but the constants give worse performance than the systems that people actually use.

7.7.2 Count-min sketches

A **count-min sketch** is used for the case where we are presented with a sequence of pairs (i_t, c_t) where $1 \le i_t \le n$ is an *index* and c_t is a *count*, and we want to construct a sketch that will allows us to approximately answer statistical queries about the vector a given by $a_i = \sum_{t,i_t=i} c_t$. These were developed by Cormode and Muthukrishnan [CM05], although some of the presentation here is based on [MU17, §15.4]. Structurally, they are related to counting Bloom filters (§7.6.5).

Note that we are no longer interested in detecting unique values, but instead want to avoid the cost of storing the entire vector a when most of its components are small or zero. The goal is that the size of the sketch should be polylogarithmic in the size of a and the length of the stream, and polynomial in the error bounds. We also want updating the sketch for each new data point to be cheap.

The Cormode-Muthukrishnan count-min sketch is fairly versatile, giving approximations of a_i, $\sum_{i=\ell}^{r} a_i$, and $a \cdot b$ (for any fixed b), and it can also be used for more complex tasks like finding **heavy hitters**—indices with high weight. The easiest case is approximating a_i when all the c_t are non-negative, so we'll start with that.

7.7.2.1 Initialization and updates

To construct a count-min sketch, build a two-dimensional array c with depth $d = \lceil \ln(1/\delta) \rceil$ and width $w = \lceil e/\epsilon \rceil$, where ϵ is the error bound and δ is the probability of exceeding the error bound. Choose d independent hash functions from some 2-universal hash family; we'll use one of these hash function for each row of the array. Initialize c to all zeros.

The update rule: Given an update (i_t, c_t), increment $c[j, h_j(i_t)]$ by c_t for $j = 1 \dots d$. (This is the *count* part of count-min.)

7.7.2.2 Queries

Let's start with **point queries**. Here we want to estimate a_i for some fixed i. There are two cases; the first handles non-negative increments only, while the second handles arbitrary increments. In both cases we will get an estimate whose error is linear in both the error parameter ϵ and the ℓ_1-norm $\|a\|_1 = \sum_i |a_i|$ of a. It follows that the relative error will be low for heavy points, but we may get a large relative error for light points (and especially large for points that don't appear in the data set at all).

For the non-negative case, to estimate a_i, compute $\hat{a}_i = \min_j c[j, h_j(i)]$. (This is the *min* part of coin-min.) Then:

Lemma 7.7.1. *When all c_t are non-negative, for $\hat{a}_i$ as defined above:*

$$\hat{a}_i \geq a_i, \tag{7.7.1}$$

and

$$\Pr\left[\hat{a}_i \leq a_i + \epsilon \|a\|_1\right] \geq 1 - \delta. \tag{7.7.2}$$

Proof. The lower bound is easy. Since for each pair (i, c_t) we increment each $c[j, h_j(i)]$ by c_t, we have an invariant that $a_i \leq c[j, h_j(i)]$ for all j throughout the computation, which gives $a_i \leq \hat{a}_i = \min_j c[j, h_j(i)]$.

For the upper bound, let I_{ijk} be the indicator for the event that $(i \neq k) \wedge (h_j(i) = h_j(k))$, i.e., that we get a collision between i and k using h_j. The 2-universality property of the h_j gives $\mathrm{E}[I_{ijk}] \leq 1/w \leq \epsilon/e$.

Now let $X_{ij} = \sum_{k=1}^{n} I_{ijk} a_k$. Then $c[j, h_j(i)] = a_i + X_{ij}$. (The fact that $X_{ij} \geq 0$ gives an alternate proof of the lower bound.) Now use linearity of

expectation to get

$$\begin{aligned}
\mathrm{E}\left[X_{ij}\right] &= \mathrm{E}\left[\sum_{k=1}^{n} I_{ijk} a_k\right] \\
&= \sum_{k=1}^{n} a_k \mathrm{E}\left[I_{ijk}\right] \\
&\leq \sum_{k=1}^{n} a_k(\epsilon/e) \\
&= (\epsilon/e)\|a\|_1.
\end{aligned}$$

So $\Pr\left[c[j, h_j(i)] > a_i + \epsilon\|a\|_1\right] = \Pr\left[X_{ij} > e\,\mathrm{E}\left[X_{ij}\right]\right] < 1/e$, by Markov's inequality. With d choices for j, and each h_j chosen independently, the probability that every count is too big is at most $(1/e)^d = e^{-d} \leq \exp(-\ln(1/\delta)) = \delta$. □

Now let's consider the general case, where the increments c_t might be negative. We still initialize and update the data structure as described in §7.7.2.1, but now when computing $\hat{a}_i$, we use the median count instead of the minimum count: $\hat{a}_i = \text{median}\left\{c[j, h_j(i)] \mid j = 1 \ldots n\right\}$. Now we get:

Lemma 7.7.2. *For $\hat{a}_i$ as defined above,*

$$\Pr\left[a_i - 3\epsilon\|a\|_1 \leq \hat{a}_i \leq a_i + 3\epsilon\|a\|_1\right] > 1 - \delta^{1/4}. \tag{7.7.3}$$

Proof. The basic idea is that for the median to be off by t, at least $d/2$ rows must give values that are off by t. We'll show that for $t = 3\epsilon\|a\|_1$, the expected number of rows that are off by t is at most $d/8$. Since the hash functions for the rows are chosen independently, we can use Chernoff bounds to show that with a mean of $d/8$, the chances of getting all the way to $d/2$ are small.

In detail, we again define the error term X_{ij} as above, and observe that

$$\begin{aligned}
\mathrm{E}\left[|X_{ij}|\right] &= \mathrm{E}\left[\left|\sum_{k} I_{ijk} a_k\right|\right] \\
&\leq \sum_{k=1}^{n} |a_k \mathrm{E}\left[I_{ijk}\right]| \\
&\leq \sum_{k=1}^{n} |a_k|(\epsilon/e) \\
&= (\epsilon/e)\|a\|_1.
\end{aligned}$$

Using Markov's inequality, we get $\Pr[|X_{ij}|] > 3\epsilon\|a\|_1] = \Pr[|X_{ij}| > 3e\,\mathrm{E}[X_{ij}]] < 1/3e < 1/8$. In order for the median to be off by more than $3\epsilon\|a\|_1$, we need $d/2$ of these low-probability events to occur. The expected number that occur is $\mu = d/8$, so applying the standard Chernoff bound (5.2.1) with $\delta = 3$ we are looking at

$$\begin{aligned}\Pr[S \geq d/2] &= \Pr[S \geq (1+3)\mu] \\ &\leq (e^3/4^4)^{d/8} \\ &\leq (e^{3/8}/2)^{\ln(1/\delta)} \\ &= \delta^{\ln 2 - 3/8} \\ &< \delta^{1/4}.\end{aligned}$$

(The actual exponent is about 0.31, but 1/4 is easier to deal with). This immediately gives (7.7.3). □

One way to think about this is that getting an estimate within $\epsilon\|a\|_1$ of the right value with probability at least $1-\delta$ requires 3 times the width and 4 times the depth—or 12 times the space and 4 times the time—when we aren't assuming increments are non-negative.

Next, we consider inner products. Here we want to estimate $a \cdot b$, where a and b are both stored as count-min sketches using the same hash functions. The paper concentrates on the case where a and b are both non-negative, which has applications in estimating the size of a join in a database. The method is to estimate $a \cdot b$ as $\min_j \sum_{k=1}^{w} c_a[j,k] \cdot c_b[j,k]$.

For a single j, the sum consists of both good values and bad collisions; we have $\sum_{k=1}^{w} c_a[j,k] \cdot c_b[j,k] = \sum_{k=1}^{n} a_i b_i + \sum_{p \neq q, h_j(p) = h_j(q)} a_p b_q$. The second term has expectation

$$\begin{aligned}\sum_{p \neq q} \Pr[h_j(p) = h_j(q)]\, a_p b_q &\leq \sum_{p \neq q} (\epsilon/e) a_p b_q \\ &\leq \sum_{p,q} (\epsilon/e) a_p b_q \\ &\leq (\epsilon/e)\|a\|_1 \|b\|_1.\end{aligned}$$

As in the point-query case, we get probability at most $1/e$ that a single j gives a value that is too high by more than $\epsilon\|a\|_1\|b\|_1$, so the probability that the minimum value is too high is at most $e^{-d} \leq \delta$.

7.7.2.3 Finding heavy hitters

Here we want to find the heaviest elements in the set: those indices i for which a_i exceeds $\phi\|a\|_1$ for some constant threshold $0 < \phi \leq 1$. We assume that all c_t are non-negative. Because $\|a\|_1 = \sum_i a_i$, we know that there will be at most $1/\phi$ heavy hitters. But the tricky part is figuring out which elements they are.

The output at any stage will be approximate in the following sense: it is guaranteed that any i such that $a_i \geq \phi\|a\|_1$ is included, and each i with $a_i < (\phi - \epsilon)$ that previously appeared in the stream is included with probability at most $1 - \delta$. This is similar to what we would get if we just ran a point query on all possible i, but (a) there are many possible i and (b) we won't ever output an i we've never seen.

The trick is to extend the data structure and update procedure to track all the heavy elements found so far (stored in a heap, with the minimum estimate at the top), as well as $\|a\|_1 = \sum c_t$. When a new increment (i, c) comes in, we first update the count-min structure and then do a point query on a_i; if $\hat{a}_i \geq \phi\|a\|_1$, we insert i into the heap. We also delete any elements at the top of the heap that have a point-query estimate below threshold.

Because $\hat{a}_i \geq a_i$, every heavy hitter is correctly identified. However, it's possible that an index stops being a heavy hitter at some point (because the threshold $\phi\|a\|_1$ rose since we included it). In this case it may get removed from the heap, but if it becomes a heavy hitter again, we'll put it back.

7.8 Locality-sensitive hashing

Locality-sensitive hashing was invented by Indyk and Motwani [IM98] to solve the problem of designing a data structure that finds approximate nearest neighbors to query points in high dimension. We'll mostly be following this paper in this section, concentrating on the hashing parts.

7.8.1 Approximate nearest neighbor search

In the **nearest neighbor search** problem (**NNS** for short), we are given a set of n points P in a metric space with distance function d, and we want to construct a data structure that allows us to quickly find the closet point p in P to any given query point q. We could always compute the distance between q and each possible p, but this takes time $O(n)$, and we'd like to get lookups to be sublinear in n.

Indyk and Motwani were particularly interested in what happens in $\mathbb{R}^d$ for high dimension d under various natural metrics. Because the volume of a ball in a high-dimensional space grows exponentially with the dimension, this problem suffers from the **curse of dimensionality** [Bel57]: simple techniques based on, for example, assigning points in P to nearby locations in a grid may require searching exponentially many grid locations. Indyk and Motwani deal with this through a combination of randomization and solving the weaker problem of **ϵ-nearest neighbor search** (**ϵ-NNS**), where it's acceptable to return a different point p' as long as $d(q,p') \le (1+\epsilon)\min_{p\in P} d(q,p)$.

This problem can be solved by reduction to a simpler problem called **ϵ-point location in equal balls** or **ϵ-PLEB**. In this problem, we are given n radius-r balls centered on points c in a set C, and we want a data structure that returns a point $c' \in C$ with $d(q,c') \le (1+\epsilon)r$ if there is at least one point c with $d(q,c) \le r$. If there is no such point, the data structure may or may not return a point (it might say no, or it might just return a point that is too far away, which we can discard). The difference between an ϵ-PLEB and NNS is that an ϵ-PLEB isn't picky about returning the closest point to q if there are multiple points that are all good enough. Still, we can reduce NNS to ϵ-PLEB.

The easy reduction is to use binary search. Let $R = \frac{\max_{x,y\in P} d(x,y)}{\min_{x,y\in P, x\neq y} d(x,y)}$. Given a point q, look for the minimum $\ell \in \{(1+\epsilon)^0, (1+\epsilon)^1, \ldots, R\}$ for which an ϵ-PLEB data structure with radius ℓ and centers P returns a point p with $d(q,p) \le (1+\epsilon)\ell$; then return this point as the approximate nearest neighbor.

This requires $O(\log_{1+\epsilon} R)$ instances of the ϵ-PLEB data structure and $O(\log\log_{1+\epsilon} R)$ queries. The blowup as a function of R can be avoided using a more sophisticated data structure called a **ring-cover tree**, defined in the paper. We won't talk about ring-cover trees because they are (a) complicated and (b) not randomized. Instead, we'll move directly to the question of how we solve ϵ-PLEB.

7.8.2 Locality-sensitive hash functions

Definition 7.8.1 ([IM98]). *A family of hash functions H is (r_1, r_2, p_1, p_2)-**sensitive** for d if, for any points p and q, if h is chosen uniformly from H,*

1. *If $d(p,q) \le r_1$, then $\Pr[h(p) = h(q)] \ge p_1$, and*
2. *If $d(p,q) > r_2$, then $\Pr[h(p) = h(q)] \le p_2$.*

These are useful if $p_1 > p_2$ and $r_1 < r_2$; that is, we are more likely to hash inputs together if they are closer. Ideally, we can choose r_1 and r_2 to build ϵ-PLEB data structures for a range of radii sufficient to do binary search as described above (or build a ring-cover tree if we are doing it right). For the moment, we will aim for an (r_1, r_2)-PLEB data structure, which returns a point within r_1 with high probability if one exists, and never returns a point farther away than r_2.

There is some similarity between locality-sensitive hashing and a more general dimension-reduction technique known as the **Johnson-Lindenstrauss lemma** [JL84]; this says that projecting n points in a high-dimensional space to $O(\epsilon^{-2} \log n)$ dimensions using an appropriate random matrix preserves ℓ_2 distances between the points to within relative error ϵ (in fact, even a random matrix with entries in $\{-1, 0, +1\}$ is enough [Ach03]). Unfortunately, dimension reduction by itself is not enough to solve approximate nearest neighbors in sublinear time, because we may still need to search a number of boxes exponential in $O(\epsilon^{-2} \log n)$, which will be polynomial in n. But we'll look at the Johnson-Lindenstrauss lemma and its many other applications more closely in Chapter 8.

7.8.3 Constructing an (r_1, r_2)-PLEB

The first trick is to amplify the difference between p_1 and p_2 so that we can find a point within r_1 of our query point q if one exists. This is done in three stages: First, we concatenate multiple hash functions to drive the probability that distant points hash together down until we get few collisions: the idea here is that we are taking the AND of the events that we get collisions in the original hash function. Second, we hash our query point and target points multiple times to bring the probability that nearby points hash together up: this is an OR. Finally, we iterate the procedure to drive down any remaining probability of failure below a target probability δ: another AND.

For the first stage, let $k = \log_{1/p_2} n$ and define a composite hash function $g(p) = (h_1(p) \dots h_k(p))$. If $d(p, q) > r_2$, $\Pr[g(p) = g(q)] \leq p_2^k = p_2^{\log_{1/p_2} n} = 1/n$. Adding this up over all n points in our data structure gives us one false match for q on average.

However, we may not be able to find the correct match for q, since p_1 may not be all that much larger than p_2. For this, we do a second round of amplification, where now we are taking the OR of events we want instead of the AND of events we don't want.

Let $\ell = n^\rho$, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \frac{\log p_1}{\log p_2} < 1$, and choose hash functions $g_1 \dots g_\ell$ independently as above. To store a point p, put it in a bucket for

$g_j(p)$ for each j; these buckets are themselves stored in a hash table (by hashing the value of $g_j(p)$ down further) so that they fit in $O(n)$ space. Suppose now that $d(p,q) \le r_1$ for some p. Then

$$\begin{aligned}\Pr\left[g_j(p) = g_j(q)\right] &\ge p_1^k \\ &= p_1^{\log_{1/p_2} n} \\ &= n^{-\frac{\log 1/p_1}{\log 1/p_2}} \\ &= n^{-\rho} \\ &= 1/\ell.\end{aligned}$$

So by searching through ℓ independent buckets we find p with probability at least $1 - (1 - 1/\ell)^\ell = 1 - 1/e + o(1)$. We'd like to guarantee that we only have to look at $O(n^\rho)$ points (most of which we may reject) during this process; but we can do this by stopping if we see more than 2ℓ points. Since we only expect to see ℓ bad points in all ℓ buckets, this event only happens with probability $1/2$. So even adding it to the probability of failure from the hash functions not working right we still have only a constant probability of failure $1/e + 1/2 + o(1)$.

Iterating the entire process $O(\log(1/\delta))$ times then gives the desired bound δ on the probability that this process fails to find a good point if one exists.

Multiplying out all the costs gives a cost of a query of $O(k\ell \log(1/\delta)) = O\left(n^\rho (\log_{1/p_2} n) \log(1/\delta)\right)$ hash function evaluations and $O(n^\rho \log(1/\delta))$ distance computations. The cost to insert a point is just $O(k\ell \log(1/\delta)) = O\left(n^\rho (\log_{1/p_2} n) \log(1/\delta)\right)$ hash function evaluations, the same number as for a query.

7.8.4 Hash functions for Hamming distance

Suppose that our points are d-bit vectors and that we use Hamming distance for our metric. In this case, using the family of one-bit projections $\{h_i \mid h_i(x) = x_i\}$ gives a locality-sensitive hash family [ABMRT96].

Specifically, we can show this family is $\left(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d}\right)$-sensitive. The argument is trivial: if two points p and q are at distance r or less, they differ in at most r places, so the probability that they hash together is just the probability that we don't pick one of these places, which is at least $1 - \frac{r}{d}$. Essentially the same argument works when p and q are far away.

These are not particularly clever hash functions, so the heavy lifting will be done by the (r_1, r_2)-PLEB construction. Our goal is to build an ϵ-PLEB for any fixed r, which will correspond to an $(r, r(1+\epsilon))$-PLEB. The main thing we need to do, following [IM98] as always, is compute a reasonable bound on $\rho = \frac{\log p_1}{\log p_2} = \frac{\ln(1-r/d)}{\ln(1-(1+\epsilon)r/d)}$. This is essentially just a matter of hitting it with enough inequalities, although there are a couple of tricks in the middle.

Compute

$$\begin{aligned}
\rho &= \frac{\ln(1-r/d)}{\ln(1-(1+\epsilon)r/d)} \\
&= \frac{(d/r)\ln(1-r/d)}{(d/r)\ln(1-(1+\epsilon)r/d)} \\
&= \frac{\ln((1-r/d)^{d/r})}{\ln((1-(1+\epsilon)r/d)^{d/r})} \\
&\leq \frac{\ln(e^{-1}(1-r/d))}{\ln e^{-(1+\epsilon)}} \\
&= \frac{-1+\ln(1-r/d)}{-(1+\epsilon)} \\
&= \frac{1}{1+\epsilon} - \frac{\ln(1-r/d)}{1+\epsilon}. \qquad (7.8.1)
\end{aligned}$$

Note that we used the fact that $1 + x \leq e^x$ for all x in the denominator and $(1-x)^{1/x} \geq e^{-1}(1-x)$ for $x \in [0,1]$ in the numerator. The first fact is our usual favorite inequality.

The second can be proved in a number of ways. The most visually intuitive is that $(1-x)^{1/x}$ and $e^{-1}(1-x)$ are equal at $x = 1$ and equal in the limit as x goes to 0, while $(1-x)^{1/x}$ is concave in between 0 and 1 and $e^{-1}(1-x)$ is linear. Unfortunately it is rather painful to show that $(1-x)^{1/x}$ is in fact concave. An alternative is to rewrite the inequality $(1-x)^{1-x} \geq e^{-1}(1-x)$ as $(1-x)^{1/x-1} \geq e^{-1}$, apply a change of variables $y = 1/x$ to get $(1-1/y)^{y-1} \geq e^{-1}$ for $y \in [1, \infty)$, and then argue that (a) equality holds in the limit as y goes to infinity, and (b) the left-hand-side is

a nonincreasing function, since

$$\begin{aligned}
\frac{d}{dy}\ln\left((1-1/y)^{y-1}\right) &= \frac{d}{dy}\left[(y-1)\left(\ln(y-1)-\ln y\right)\right] \\
&= \ln(1-1/y) + (y-1)\left(\frac{1}{y-1}-\frac{1}{y}\right) \\
&= \ln(1-1/y) + 1 - (1-1/y) \\
&= \ln(1-1/y) + 1/y \\
&\le -1/y + 1/y \\
&= 0.
\end{aligned}$$

We now return to (7.8.1). We'd really like the second term to be small enough that we can just write n^ρ as $n^{1/(1+\epsilon)}$. (Note that even though it looks negative, it isn't, because $\ln(1-r/d)$ is negative.) So we pull a rabbit out of a hat by assuming that $r/d < 1/\ln n$.[8] This assumption can be justified by modifying the algorithm so that d is padded out with up to $d\ln n$ unused junk bits if necessary. Using this assumption, we get

$$\begin{aligned}
n^\rho &< n^{1/(1+\epsilon)} n^{-\ln(1-1/\ln n)/(1+\epsilon)} \\
&= n^{1/(1+\epsilon)}(1-1/\ln n)^{-\ln n} \\
&\le e n^{1/(1+\epsilon)}.
\end{aligned}$$

Plugging into the formula for (r_1, r_2)-PLEB gives $O(n^{1/(1+\epsilon)}\log n \log(1/\delta))$ hash function evaluations per query, each of which costs $O(1)$ time, plus $O(n^{1/(1+\epsilon)}\log(1/\delta))$ distance computations, which will take $O(d)$ time each. If we add in the cost of the binary search, we have to multiply this by $O(\log\log_{1+\epsilon} R \log\log\log_{1+\epsilon} R)$, where the log-log-log comes from having to adjust δ so that the error doesn't accumulate too much over all $O(\log\log R)$ steps. The end result is that we can do approximate nearest-neighbor queries in

$$O\left(n^{1/(1+\epsilon)}\log(1/\delta)(\log n + d)\log\log_{1+\epsilon} R \log\log\log_{1+\epsilon} R\right)$$

time. For ϵ reasonably large, this is much better than naively testing against all points in our database, which takes $O(nd)$ time (although it does produce an exact result).

[8] Indyk and Motwani pull this rabbit out of a hat a few steps earlier, but it's pretty much the same rabbit either way.

7.8.5 Hash functions for ℓ_1 distance

Essentially the same approach works for (bounded) ℓ_1 distance, using **discretization**, where we replace a continuous variable over some range with a discrete variable. Suppose we are working in $[0,1]^d$ with the ℓ_1 metric. Represent each coordinate x_i as a sequence of d/ϵ values x_{ij} in **unary**, for $j = 1 \ldots \epsilon d$, with $x_{ij} = 1$ if $\epsilon j/d < x_i$. Then the Hamming distance between the bit-vectors representing x and y is proportional to the ℓ_1 distance between the original vectors, plus an error term that is bounded by ϵ. We can then use the hash functions for Hamming distance to get a locality-sensitive hash family.

A nice bit about this construction is that we don't actually have to build the bit-vectors; instead, we can specify a coordinate x_i and a threshold c and get the same effect by recording whether $x_i > c$ or not.

Note that this does increase the cost slightly: we are converting d-dimensional vectors into (d/ϵ)-long bit vectors, so the $\log(n+d)$ term becomes $\log(n+d/\epsilon)$. When n is small, this effectively multiples the cost of a query by an extra $\log(1/\epsilon)$. More significant is that we have to cut ϵ in half to obtain the same error bounds, because we now pay ϵ error for the data structure itself and an additional ϵ error for the discretization. So our revised cost for the ℓ_1 case is

$$O\left(n^{1/(1+\epsilon/2)} \log(1/\delta)(\log n + d/\epsilon) \log\log_{1+\epsilon/2} R \log\log\log_{1+\epsilon/2} R\right).$$

Chapter 8

Dimension reduction

In this chapter, we will discuss how randomization can be used to reduce the dimension of a set of points in a way that approximately preserves the distance between points. The main tool for doing this is a family of closely-related results that collectively are known as the Johnson-Lindenstrauss lemma.

We can't really do full justice to the Johnson-Lindenstrauss lemma and its applications here, although we will try to hit the high points. There is an excellent survey by Freksen [Fre21] if you'd like to learn more.

8.1 The Johnson-Lindenstrauss lemma

The **Johnson-Lindenstrauss lemma** [JL84] says that it is possible to project a set of n vectors in a space of arbitrarily high dimension onto an $O(\log n)$-dimensional subspace, such that the distances between the vectors are approximately preserved. There are several versions of the lemma, depending on how the projection is done, but in each case we want to show that for sufficiently large k as a function of n there is some $k \times d$ matrix A such that for any two d-dimensional vectors u and v in our set, $(1-\epsilon)\|u-v\|^2 \leq \|Au - Av\|^2 \leq (1+\epsilon)\|u-v\|^2$. Typical choices for A are:

- A projection matrix for a uniform random k-dimensional subspace. (Original Johnson-Lindenstrauss paper [JL84], also used in the Dasgupta-Gupta [DG03] proof described below in §8.1.2.)
- A matrix whose elements are i.i.d. normal random variables. (Indyk-Motwani [IM98].)

- A matrix whose elements are i.i.d. variables with a particular distribution on $\{-1, 0, +1\}$. (Achlioptas [Ach03].)
- A matrix whose elements are ± 1 coin-flips [Ach03]. This is perhaps the easiest to keep track of, but the constants in the error bounds are a bit worse than the $\{-1, 0, +1\}$ version.

In each case, we multiply the matrix by an appropriate fixed scale factor c so that $\mathrm{E}\left[\|cAu\|^2\right] = \|u\|^2$. The resulting linear projection is known as a **Johnson-Lindenstrauss transformation** (**JLT** for short).

8.1.1 Reduction to single-vector case

Most proofs of the theorem reduce to the case of a single vector, by finding a value of k such that $\Pr\left[(1-\epsilon)\|u\|^2 \leq \|Au\|^2 \leq (1+\epsilon)\|u\|^2\right] \geq 1 - 2/n^2$, and then using the union bound to show that this same property holds for all $\binom{n}{2}$ vectors $u - v$ with nonzero probability. This shows the existence of a good matrix, and we can generate matrices and test them until we find one that actually works.

8.1.2 A relatively simple proof of the lemma

This proof is due to Dasgupta and Gupta [DG03], and is somewhat simpler than many proofs of the theorem. The basic idea is that if A projects onto a uniform random k-dimensional subspace, then its effect on the length of an arbitrary nonzero vector u is the same as its effect on the length of a unit vector $u/\|u\|$, which is the same as the effect of some specific fixed $k \times d$ matrix B applied to a random unit vector Y drawn uniformly from the surface of the d-dimensional sphere S^d. An easy choice for B is just the matrix that extracts the first k coordinates from Y.

So how do we get a random unit vector Y? The usual trick is to use the fact that the multivariate normal distribution is radially symmetric: if we generate d independent normally-distributed random variables $X_1, \ldots, X_d$, then the vector $Y = X/\|X\|$ is uniformly distributed over S^d.[1]

This gives $Z = BY = (X_1, \ldots, X_k)/\|X\|$, and $\|Z\|^2 = (X_1^2 + X_2^2 + \cdots + X_k^2)/(X_1^2 + \cdots + X_d^2)$. If each value X_i^2 were exactly equal to its expectation

[1] Radial symmetry is immediate from the density $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ of the univariate normal distribution. If we consider a vector $\langle X_1, \ldots, X_d \rangle$ of independent $N(0,1)$ random variables, then the joint density is given by the product $\prod_{i=1}^{d} \frac{1}{2\pi} e^{-x_i^2/2} = (2\pi)^{-d/2} e^{-\sum x_i^2/2}$. But $\sum x_i^2 = r^2$ where r is the distance from the origin, meaning this distribution has the same density at all points at the same distance.

1 (the variance of a standard $N(0,1)$ normal random variable), then $\|Z\|^2$ would be exactly equal to k/d, and we could adjust the length of Z to equal 1 by multiplying Z by a correction factor of $\sqrt{d/k}$. But we generally won't have each X_i^2 equal to 1; instead, we will use a Chernoff bound argument to show that $\|Z\|^2$ lies between $(1-\epsilon)(k/d)$ and $(1+\epsilon)(k/d)$, for suitable ϵ and k, with high probability.

Following the approach in [DG03], we'll write β for $(1-\epsilon)$ or $(1+\epsilon)$ as appropriate. Starting with the lower bound, we want to show that for $\beta = 1-\epsilon$, it is unlikely that

$$\|Z\|^2 \leq \beta(k/d)$$

which expands to

$$\frac{\sum_{i=1}^k X_i^2}{\sum_{i=1}^d X_i^2} \leq \beta(k/d).$$

Having a ratio between two sums is a nuisance, but we can multiply out the denominators to turn it into something we can apply a Chernoff-style argument to.

$$\begin{aligned}
\Pr\left[\frac{\sum_{i=1}^k X_i^2}{\sum_{i=1}^d X_i^2} \leq \beta(k/d)\right] &= \Pr\left[d\sum_{i=1}^k X_i^2 \leq \beta k \sum_{i=1}^d X_i^2\right] \\
&= \Pr\left[\beta k(X_1^2 + \cdots + X_d^2) - d(X_1^2 + \ldots X_k^2) \geq 0\right] \\
&= \Pr\left[\exp\Big(t\Big(\beta k(X_1^2 + \cdots + X_d^2) - d(X_1^2 + \ldots X_k^2)\Big)\Big) \geq 1\right] \\
&\leq \mathrm{E}\left[\exp(t(\beta k(X_1^2 + \cdots + X_d^2) - d(X_1^2 + \ldots X_k^2)))\right] \\
&= \mathrm{E}\left[\exp(t\beta k X^2)\right]^{d-k} \mathrm{E}\left[\exp(t(\beta k - d)X^2)\right]^k,
\end{aligned}$$

where t can be any value greater than 0, the shift from probability to expectation uses Markov's inequality, and in the last step we replace each independent occurrence of X_i with a standard normal random variable X.

Now we just need to be able to compute the moment generating function $\mathrm{E}\left[e^{sX^2}\right]$ for X^2. The quick way to do this is to notice that X^2 has a chi-squared distribution with one degree of freedom (since the chi-squared distribution with k degrees of freedom is just the distribution of the sum of squares of k independent normal random variables), and look up its m.g.f. $(1-2s)^{-1/2}$ (for $s < 1/2$).

We can substitute this m.g.f. in the formula above to get

$$\Pr\left[(X_1^2+\cdots+X_k^2)/(X_1^2+\cdots+X_d^2) \le \beta(k/d)\right] \le \left(\frac{1}{1-2\beta kt}\right)^{(d-k)/2}\left(\frac{1}{2t(\beta k-d)}\right)^{k/2}.$$

The rest is just the usual trick of finding the minimum value of this expression over all values t (that don't produce negative denominators in the m.g.f.!) by differentiating and setting to 0. This turns out to be easiest if we maximize $1/g(t)^2$ where $g(t)$ is the above expression; see the Dasgupta-Gupta paper for details. For $\beta = 1-\epsilon < 1$, they show that the optimal t is $\frac{1-\beta}{2\beta(d-k\beta)}$, which gives, after a few intermediate steps,

$$\Pr\left[\|Z\|^2 \le \beta(k/d)\right] \le e^{(k/2)(1-\beta+\ln\beta)}.$$

The same argument applied to $g(-t)$ gives essentially the same bound for $\beta = 1+\epsilon > 1$:

$$\Pr\left[\|Z\|^2 \ge \beta(k/d)\right] \le e^{(k/2)(1-\beta+\ln\beta)}.$$

The $1-\beta+\ln\beta$ factors are very close to 0. For $\beta = 1-\epsilon$, we have $1-\beta+\ln\beta = \epsilon+\ln(1-\epsilon) = \Theta(-\epsilon^2)$, and for $\beta = 1+\epsilon$, we similarly have $1-\beta+\ln\beta = -\epsilon+\ln(1+\epsilon) = \Theta(-\epsilon^2)$. So in either case we need $k = \Theta(\epsilon^{-2}\ln n)$ to make the probability bound polynomially small in n.

A more precise calculation (see [DG03]) includes the next term in the Taylor series expansion of $1\pm\epsilon$ to get an exact inequality. This gives a good map at $k \ge 4(\epsilon^2/2-\epsilon^3/3)^{-1}\ln n$. Note that both our crummy asymptotic bound and this more precise bound depend on n but not d.

Constructing a projection matrix for a random k-dimensional space is mildly painful. The easiest way to do it may be to generate a random $k\times d$ matrix of independent $N(0,1)$ variables and then apply Gram-Schmidt orthogonalization, which takes $O(k^2d)$ time. A faster approach that gives similar results is to use a matrix of independent random variables that are ±1 with probability $1/6$ each and 0 the rest of the time. This doesn't produce exactly the same distribution on projections, but it can be shown (with much more effort) to still work pretty well [Ach03].

If we leave out enough details, we can summarize all of these results as a single lemma:

Lemma 8.1.1 ([JL84]). *For every set of n points X in $\mathbb{R}^d$, and any ϵ with $0<\epsilon<1$, there is a linear projection $f:\mathbb{R}^d\to\mathbb{R}^k$ with $k = O(\epsilon^{-2}\log n)$ such that for any u and v in X,*

$$(1-\epsilon)\|u-v\|^2 \le \|f(u)-f(v)\| \le (1+\epsilon)\|u-v\|.$$

8.1.3 Distributional version

It's worth nothing that the argument for Lemma 8.1.1 doesn't use any property of X except its size. This means that the same argument works (with nonzero probability) without needing to know X. This gives the "distributional" version of the lemma, which says

Lemma 8.1.2 ([JL84])**.** *For every d, $0 < \epsilon < 1$, and $0 < \delta < 1$, there exists a distribution over linear functions $f : \mathbb{R}^d \to R^k$ with $k = O(\epsilon^{-2} \log(1/\delta))$ such that for every $x \in R^d$,*

$$\Pr\left[(1-\epsilon)\|x\|^2 \leq \|f(x)\|^2 \leq (1+\epsilon)\|x\|^2\right] \geq 1 - \delta.$$

This can be handy for applications where we don't know the vectors we will be working with in advance.

8.2 Applications

The intuition is that the Johnson-Lindenstrauss lemma lets us reduce the dimension of some problem involving distances between n points, where we are willing to tolerate a small constant relative error, from some arbitrary d to $O(\log n)$ (or $O(\log(1/\delta))$ if we just care about the error probability per pair of points). So applications tend to fall into one of two categories:

1. We want to run some algorithm on a set of points in a high-dimensional space, but the cost of the algorithm is high (perhaps exponential!) as a function of the dimension.

2. We want to run some algorithm on a set of points in a high-dimensional space, but we don't want to pay the linear-in-the-dimension space costs.

If we are lucky, we'll get both payoffs, winning on both time and space.

For example, suppose we have a set of n points $x_1, x_2, \ldots, x_n$ representing the centers of various clusters in a d-dimensional space, and we want to rapidly classify incoming points y into one of these clusters by finding which x_i y is closest to. If we do this naively, this is a $\Theta(nd)$ operation, since it takes $\Theta(d)$ time to compute each distance between y and some x_i. If instead we are willing to accept the inaccuracy associated with Johnson-Lindenstrauss, we can fix a matrix A in advance, replace each x_i with Ax_i, and find the x_i that is (approximately) closest to y using $O(d \log n)$ time to reduce y to Ay and $O(n \log n)$ time to compute the distance between Ay and each Ax_i. In this case we are reducing both the time complexity of our classification

algorithm (at least if we don't count the pre-processing time to generate the Ax_i) and the amount of data we need to store.

An example of space savings is the use of the Johnson-Lindenstrauss transform in streaming algorithms (see §7.7). Freksen [Fre21] gives a simple example of estimating $\|x\|^2$ where x is a vector of counts of items from a set of size n presented one at a time. If we don't charge for the space to store the JLT function f, we can simply add the i-th column of the matrix to our running total whenever we see item i, and we need only store $O(\epsilon^{-2}\log(1/\delta))$ distinct numerical values of an appropriate precision to estimate $\|x\|^2$ to within ϵ relative error with probability at least $1-\delta$. The problem is that in reality we do need to represent f somehow, and even for a ± 1 matrix this will take $\Theta(n\epsilon^{-2}\log(1/\delta))$ space. Fortunately it can be shown that generating f using a 4-independent hash function reduces the space for f to $O(\log n)$, giving the **Tug-of-War sketch** of Alon *et al.* [AMS96], one of the first compact streaming data structures. Though this is a nice application of the JLT, it's worth mentioning Cormode and Muthukrishnan [CM05] observe that this is still significantly more costly for most queries than their own count-min sketch.

Chapter 9

Martingales and stopping times

In §5.3.2, we used martingales to show that the outcome of some process was tightly concentrated. Here we will show how martingales interact with **stopping times**, which are random variables that control when we stop carrying out some task. This will require a few new definitions.

9.1 Definitions

The general form of a martingale $\{X_t, \mathcal{F}_t\}$ consists of:

- A sequence of random variables $X_0, X_1, X_2, \ldots$; and
- A **filtration** $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \ldots$, where each σ-algebra $\mathcal{F}_t$ represents our knowledge at time t;

subject to the requirements that:

1. The sequence of random variables is **adapted** to the filtration, which just means that each X_t is $\mathcal{F}_t$-measurable or equivalently that $\mathcal{F}_t$ (and thus all subsequent $\mathcal{F}_{t'}$ for $t' \geq t$) includes all knowledge of X_t; and
2. The **martingale property**

$$\mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] = X_t \tag{9.1.1}$$

holds for all t.

We will also also need the following definition of a **stopping time**. Given a filtration $\{\mathcal{F}_t\}$, a random variable τ is a stopping time for $\{\mathcal{F}_t\}$ if $\tau \in \mathbb{N} \cup \{\infty\}$ and the event $[\tau \leq t]$ is $\mathcal{F}_t$-measurable for all $t \in \mathbb{N}$.[1] In simple terms, τ is a stopping time if you know at time t whether to stop there or not.

What we like about martingales is that iterating the martingale property shows that $\mathrm{E}[X_t] = \mathrm{E}[X_0]$ for all fixed t. We will show that, under reasonable conditions, the same holds for X_τ when τ is a stopping time. (The random variable X_τ is defined in the obvious way, as a random variable that takes on the value of X_t when $\tau = t$.)

9.2 Submartingales and supermartingales

In some cases we have a process where instead of getting equality in (9.1.1), we get an inequality instead. A **submartingale** replaces (9.1.1) with

$$X_t \leq \mathrm{E}[X_{t+1} \mid \mathcal{F}_t] \tag{9.2.1}$$

while a **supermartingale** satisfies

$$X_t \geq \mathrm{E}[X_{t+1} \mid \mathcal{F}_t]. \tag{9.2.2}$$

In each case, what is "sub" or "super" is the value at the current time compared to the expected value at the next time. Intuitively, a submartingale corresponds to a process where you win on average, while a supermartingale is a process where you lose on average. Casino games (in profitable casinos) are submartingales for the house and supermartingales for the player.

Sub- and supermartingales can be reduced to martingales by subtracting off the expected change at each step. For example, if $\{X_t\}$ is a submartingale with respect to $\{\mathcal{F}_t\}$, then the process $\{Y_t\}$ defined recursively by

$$\begin{aligned} Y_0 &= X_0 \\ Y_{t+1} &= Y_t + X_{t+1} - \mathrm{E}[X_{t+1} \mid \mathcal{F}_t] \end{aligned}$$

[1] Different authors impose different conditions on the range of τ; for example, Mitzenmacher and Upfal [MU17] exclude the case $\tau = \infty$. We allow $\tau = \infty$ to represent the outcome where we never stop. This can be handy for modeling processes where this outcome is possible, although in practice we will typically insist that it occurs only with probability zero.

is a martingale, since

$$\begin{aligned}\mathrm{E}\left[Y_{t+1} \mid \mathcal{F}_t\right] &= \mathrm{E}\left[Y_t + X_{t+1} - \mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] \mid \mathcal{F}_t\right] \\ &= Y_t + \mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] - \mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] \\ &= Y_t.\end{aligned}$$

One way to think of this is that $Y_t = X_t + \Delta_t$, where Δ_t is a predictable, non-decreasing **drift process** that starts at 0. For supermartingales, the same result holds, but now Δ_t is non-increasing. This ability to decompose an adapted stochastic process into the sum of a martingale and a predictable drift process is known as the **Doob decomposition theorem**.

9.3 The optional stopping theorem

If $(X_t, \mathcal{F}_t)$ is a martingale, then applying induction to the martingale property shows that $\mathrm{E}\left[X_t\right] = \mathrm{E}\left[X_0\right]$ for any fixed time t. The **optional stopping theorem** shows that this also happens for X_τ when τ is a stopping time, under various choices of additional conditions:

Theorem 9.3.1. *Let $(X_t, \mathcal{F}_t)$ be a martingale and τ a stopping time for $\{\mathcal{F}_t\}$. Then $\mathrm{E}\left[X_\tau\right] = E[X_0]$ if at least one of the following conditions holds:*

1. ***Bounded time.*** *There is a fixed n such that $\tau \leq n$ always.*
2. ***Finite time and bounded range.*** $\Pr\left[\tau < \infty\right] = 1$, *and there is a fixed M such that for all $t \leq \tau$, $|X_t| \leq M$.*
3. ***Finite expected time and bounded increments.*** $\mathrm{E}\left[\tau\right] < \infty$, *and there is a fixed c such that $|X_{t+1} - X_t| \leq c$ for all $t < \tau$.*
4. ***General case.*** *All three of the following conditions hold:*
   (a) $\Pr\left[\tau < \infty\right] = 1$,
   (b) $\mathrm{E}\left[|X_\tau|\right] < \infty$, *and*
   (c) $\lim_{t\to\infty} \mathrm{E}\left[X_t \cdot 1_{[\tau > t]}\right] = 0$.

It would be nice if we could show $\mathrm{E}\left[X_\tau\right] = E[X_0]$ without the side conditions, but in general this isn't true. For example, the double-after-losing martingale strategy in the St. Petersburg paradox (see §3.4.1.1) eventually yields +1 with probability 1, so if τ is the time we stop playing, we have $\Pr\left[\tau < \infty\right] = 1$, $\mathrm{E}\left[|X_\tau|\right] < \infty$, but $\mathrm{E}\left[X_\tau\right] = 1 \neq \mathrm{E}\left[X_0\right] = 0$. To make this

happen, we have to violate all of bounded time (τ is not bounded), bounded range ($|X_t|$ roughly doubles every step until we stop), bounded increments ($|X_{t+1} - X_t|$ doubles every step as well), and at least one of the three conditions of the general case (the last one: $\lim_{t\to\infty} \mathrm{E}[X_t] \cdot 1_{[\tau>t]} = -1 \neq 0$).

To prove Theorem 9.3.1, we'll use a truncation argument. The intuition is that for any fixed n, we can **truncate** X_τ to $X_{\min(\tau,n)}$ and show that $\mathrm{E}\left[X_{\min(\tau,n)}\right] = \mathrm{E}[X_0]$. This will immediately give the bounded time case. For the other cases, the argument is that $\lim_{n\to\infty} \mathrm{E}\left[X_{\min(\tau,n)}\right]$ converges to $\mathrm{E}[X_\tau]$ provided the missing part $\mathrm{E}\left[X_\tau - X_{\min(\tau,n)}\right]$ converges to zero. How we do this depends on which assumptions we are making.

We'll start by formalizing the core truncation argument:

Lemma 9.3.2. *Let $(X_t, \mathcal{F}_t)$ be a martingale and τ a stopping time for $\{\mathcal{F}_t\}$. Then for any $n \in \mathbb{N}$, $\mathrm{E}\left[X_{\min(\tau,n)}\right] = E[X_0]$.*

Proof. Define $Y_t = X_0 + \sum_{i=1}^{t}(X_t - X_{t-1})1_{[\tau>t-1]}$. Then $(Y_t, \mathcal{F}_t)$ is a martingale, because we can calculate $\mathrm{E}[Y_{t+1} \mid \mathcal{F}_t] = \mathrm{E}\left[Y_t + (X_{t+1} - X_t)1_{[\tau>t]} \;\middle|\; \mathcal{F}_t\right] = Y_t + 1_{[\tau>t]} \cdot \mathrm{E}[X_{t+1} - X_t \mid \mathcal{F}_t] = Y_t$; effectively, we are treating $1_{[\tau\leq t-1]}$ as a sequence of bets, and we know that adjusting our bets doesn't change the martingale property. But then $\mathrm{E}\left[X_{\min(\tau,n)}\right] = \mathrm{E}[Y_n] = \mathrm{E}[Y_0] = \mathrm{E}[X_0]$. □

As claimed, this gives us the bounded-time variant for free. If $\tau \leq n$ always, then $X_\tau = X_{\min(\tau,n)}$, and $\mathrm{E}[X_\tau] = \mathrm{E}\left[X_{\min(\tau,n)}\right] = \mathrm{E}[X_0]$.

For each of the unbounded-time variants, we will apply some version of the following strategy:

1. Observe that since $\mathrm{E}\left[X_{\min(\tau,n)}\right] = \mathrm{E}[X_0]$ is a constant for any fixed n, $\lim_{n\to\infty} \mathrm{E}\left[X_{\min(\tau,n)}\right]$ converges to $\mathrm{E}[X_0]$.

2. Argue using whatever assumptions we are making that $\lim_{n\to\infty} \mathrm{E}\left[X_{\min(\tau,n)}\right]$ also converges to $\mathrm{E}[X_\tau]$.

3. Conclude that $\mathrm{E}[X_0] = \mathrm{E}[X_\tau]$, since they are both limits of the same sequence.

For the middle step, start with

$$X_\tau = X_{\min(\tau,n)} + 1_{[\tau>n]}(X_\tau - X_n).$$

This holds because either $\tau \leq n$, and we just get X_τ, or $\tau > n$, and we get $X_n + (X_\tau - X_n) = X_\tau$.

Taking the expectation of both sides gives

$$\begin{aligned}\mathrm{E}\left[X_\tau\right] &= \mathrm{E}\left[X_{\min(\tau,n)}\right] + \mathrm{E}\left[1_{[\tau>n]}(X_\tau - X_n)\right] \\ &= \mathrm{E}\left[X_0\right] + \mathrm{E}\left[1_{[\tau>n]}(X_\tau - X_n)\right].\end{aligned}$$

So if we can show that the right-hand term goes to zero in the limit, we are done.

For the bounded-range case, we have $|X_\tau - X_n| \leq 2M$, so $\left|\mathrm{E}\left[1_{[\tau>n]}\right](X_\tau - X_n)\right| \leq 2M \cdot \Pr\left[\tau > n\right]$. Since in this case we assume $\Pr\left[\tau < \infty\right] = 1$, $\lim_{n\to\infty} \Pr\left[\tau > n\right] = 0$, and the theorem holds.

For bounded increments, we have

$$\begin{aligned}\left|\mathrm{E}\left[(X_\tau - X_n)1_{[\tau>n]}\right]\right| &= \left|\mathrm{E}\left[\sum_{t\geq n}(X_{t+1} - X_t)1_{[\tau>t]}\right]\right| \\ &\leq \mathrm{E}\left[\sum_{t\geq n}|(X_{t+1} - X_t)| \cdot 1_{[\tau>t]}\right] \\ &\leq \mathrm{E}\left[\sum_{t\geq n} c \cdot 1_{[\tau>t]}\right] \\ &\leq c\,\mathrm{E}\left[\sum_{t\geq n} 1_{[\tau>t]}\right].\end{aligned}$$

But $\mathrm{E}\left[\tau\right] = \sum_{t=0}^{\infty} 1_{[\tau>t]}$. Under the assumption that this sequence converges, its tail goes to zero, and again the theorem holds.

For the general case, we can expand

$$\mathrm{E}\left[X_\tau\right] = \mathrm{E}\left[X_{\min(\tau,n)}\right] + \mathrm{E}\left[1_{[\tau>n]}X_\tau\right] - \mathrm{E}\left[1_{[\tau>n]}X_n\right]$$

which implies

$$\lim_{n\to\infty} \mathrm{E}\left[X_\tau\right] = \lim_{n\to\infty} \mathrm{E}\left[X_{\min(\tau,n)}\right] + \lim_{n\to\infty} \mathrm{E}\left[1_{[\tau>n]}X_\tau\right] - \lim_{n\to\infty} \mathrm{E}\left[1_{[\tau>n]}X_n\right],$$

assuming all these limits exist and are finite. We've already established that the first limit is $\mathrm{E}\left[X_0\right]$, which is exactly what we want. So we just need to show that the other two limits both converge to zero. For the last limit, we just use condition (4c), which gives $\lim_{n\to\infty} \mathrm{E}\left[1_{[\tau>n]}X_n\right] = 0$; no further

argument is needed. But we still need to show that the middle limit also vanishes.

Here we use condition (4b). Observe that $\mathrm{E}\left[1_{[\tau>n]}X_\tau\right] = \sum_{t=n+1}^{\infty} \mathrm{E}\left[1_{[\tau=t]}X_t\right]$. Compare this with $\mathrm{E}[X_\tau] = \sum_{t=0}^{\infty} \mathrm{E}\left[1_{[\tau=t]}X_t\right]$; this is an absolutely convergent series (this is why we need condition (4b)), so in the limit the sum of the terms for $i = 0 \ldots n$ converges to $\mathrm{E}[X_\tau]$. But this means that the sum of the remaining terms for $i = n+1 \ldots \infty$ converges to zero. So the middle term goes to zero as n goes to infinity. This completes the proof.

9.4 Applications

Here we give some example of the Optional Stopping Theorem in action. In each case, the trick is to find an appropriate martingale and stopping time, and let the theorem do all the work.

9.4.1 Random walks

Let X_t be an **unbiased** ± 1 **random walk** that starts at 0, adds ± 1 to its current position with equal probability at each step, and stops if it reaches $-a$ or $+b$.[2] We'd like to calculate the probability of reaching $+b$ before $-a$. Let τ be the time at which the process stops.

We can easily show that $\Pr[\tau < \infty] = 1$ and $\mathrm{E}[\tau] < \infty$ by observing that from any state of the random walk, there is a probability of at least $2^{-(a+b)}$ that it stops within $a+b$ steps (by flipping heads $a+b$ times in a row), so that if we consider a sequence of intervals of length $a+b$, the expected number of such intervals we can have before we stop is at most 2^{a+b}, giving $\mathrm{E}[\tau] \leq (a+b)2^{a+b}$ (we can do better than this).

We also have bounded increments by the definition of the process (bounded range also works, at least up until time τ). So $\mathrm{E}[X_\tau] = E[X_0] = 0$ and the probability p of landing on $+b$ instead of $-a$ must satisfy $pb - (1-p)a = 0$, giving $p = \frac{a}{a+b}$.

Now suppose we want to find $\mathrm{E}[\tau]$. Let $Y_t = X_t^2 - t$. Then $Y_{t+1} = (X_t \pm 1)^2 - (t+1) = X_t^2 \pm 2X_t + 1 - (t+1) = (X_t^2 - t) \pm 2X_t = Y_t \pm 2X_t$. Since the plus and minus cases are equally likely, they cancel out in expectation and $\mathrm{E}[Y_{t+1} \mid \mathcal{F}_t] = Y_t$: we just showed Y_t is a martingale.[3] We can also show

[2] This is called a **random walk with two absorbing barriers**.

[3] This construction generalizes in a nice way to arbitrary martingales. Suppose $\{X_t\}$ is a martingale with respect to $\{\mathcal{F}_t\}$. Let $\Delta_t = X_t - X_{t-1}$, and let $V_t = \mathrm{Var}[\Delta_t \mid \mathcal{F}_{t-1}]$ be the conditional variance of the t-th increment (note that this is a random variable that

it has bounded increments (at least up until time τ), because $|Y_{t+1} - Y_t| = 2|X_t| \leq \max(a, b)$.

From Theorem 9.3.1, $\mathrm{E}[Y_\tau] = 0$, which gives $\mathrm{E}[\tau] = \mathrm{E}\left[X_\tau^2\right]$. But we can calculate $\mathrm{E}\left[X_\tau^2\right]$: it is $a^2 \Pr[X_\tau = -a] + b^2 \Pr[X_t = b] = a^2(b/(a+b)) + b^2(a/(a+b)) = (a^2 b + b^2 a)/(a+b) = ab$.

If we have a random walk that only stops at $+b$,[4] then if τ is the first time at which $X_\tau = b$, τ is a stopping time. However, in this case $\mathrm{E}[X_\tau] = b \neq \mathrm{E}[X_0] = 0$. So the optional stopping theorem doesn't apply in this case. But we have bounded increments, so Theorem 9.3.1 would apply if $\mathrm{E}[\tau] < \infty$. It follows that the expected time until we reach b is unbounded, either because sometimes we never reach b, or because we always reach b but sometimes it takes a very long time. [5]

---

may depend on previous outcomes). We can easily show that $Y_t = X_t^2 - \sum_{i=1}^t V_i$ is a martingale. The proof is that

$$\begin{aligned}
\mathrm{E}[Y_t \mid \mathcal{F}_{t-1}] &= \mathrm{E}\left[X_t^2 - \sum_{i=1}^t V_t \,\middle|\, \mathcal{F}_{t-1}\right] \\
&= \mathrm{E}\left[(X_{t-1} + \Delta_t)^2 \mid \mathcal{F}_{t-1}\right] - \sum_{i=1}^t V_i \\
&= \mathrm{E}\left[X_{t-1}^2 + 2X_{t-1}\Delta_t + \Delta_t^2 \mid \mathcal{F}_{t-1}\right] - \sum_{i=1}^t V_i \\
&= X_{t-1}^2 + 2X_{t-1}\,\mathrm{E}[\Delta_t \mid \mathcal{F}_{t-1}] + \mathrm{E}\left[\Delta_t^2 \mid \mathcal{F}_{t-1}\right] - \sum_{i=1}^t V_i \\
&= X_{t-1}^2 + 0 + V_t - \sum_{i=1}^t V_i \\
&= X_{t-1}^2 - \sum_{i=1}^{t-1} V_i \\
&= Y_{t-1}.
\end{aligned}$$

For the ± 1 random walk case, we have $V_t = 1$ always, giving $\sum_{i=1}^t V_i = t$ and $\mathrm{E}\left[X_\tau^2\right] = \mathrm{E}\left[X_0^2\right] + \mathrm{E}[\tau]$ when τ is a stopping time satisfying the conditions of the Optional Stopping Theorem. For the general case, the same argument gives $\mathrm{E}\left[X_\tau^2\right] = \mathrm{E}\left[X_0^2\right] + \mathrm{E}\left[\sum_{t=1}^\tau V_t\right]$ instead: the expected square position of X_t is incremented by the conditional variance at each step.

[4] This would be a **random walk with one absorbing barrier**.

[5] In fact, we always reach b. An easy way to see this is to imagine a sequence of intervals of length $n_1, n_2, \ldots$, where $n_{i+1} = \left(b + \sum_{j=1}^i n_j\right)^2$. At the end of the i-th interval, we are no lower than $-\sum_{j=0}^i n_j$, so we only need to go up $\sqrt{n_{i+1}}$ positions to reach a by the

We can also consider a **biased random walk** where $+1$ occurs with probability p and -1 with probability $q = 1 - p$. If X_t is the position of the random walk at time t, and $\mathcal{F}_t$ is the associated σ-algebra, then X_t isn't a martingale with respect to $\mathcal{F}_t$. But there are at least two ways to turn it into one:

1. Define $Y_t = X_t - (p-q)t$. Then

$$\begin{aligned}
\mathrm{E}\left[Y_{t+1} \mid \mathcal{F}_t\right] &= \mathrm{E}\left[X_{t+1} - (p-q)(t+1) \mid \mathcal{F}_t\right] \\
&= p(X_t + 1)\mathcal{F}_t + q(X_t - 1)\mathcal{F}_t - (p-q)(t+1) \\
&= (p+q)X_t + (p-q) - (p-q)(t+1) \\
&= X_t - (p-q)t \\
&= Y_t,
\end{aligned}$$

   and Y_t is a martingale with respect to $\mathcal{F}_t$.

2. Define $Z_t = (q/p)^{X_t}$. Then

$$\begin{aligned}
\mathrm{E}\left[Z_{t+1} \mid \mathcal{F}_t\right] &= p(q/p)^{X_t+1} + q(q/p)^{X_t-1} \\
&= (q/p)^{X_t}(p(q/p) + q(p/q)) \\
&= (q/p)^{X_t}(q+p) \\
&= (q/p)^{X_t} \\
&= Z_t.
\end{aligned}$$

   Again we have a martingale with respect to $\mathcal{F}_t$.

Now let's see what we can do with the Optional Stopping Theorem.

1. Suppose we start with $X_0 = 0$ and we want to know the probability p_b that we will reach $+b$ before we reach $-a$.

   Let τ be the first time at which $X_\tau \in \{-a, b\}$. We can use the same argument as in the unbiased case to show that $\Pr[\tau < \infty] = 1$ and $\mathrm{E}[\tau] < \infty$, because from any position X_t there is at least a $p^{a+b} > 0$ chance that the next $a + b$ steps will all be $+1$ and we will reach b. Since we can flip this p^{a+b}-probability coin independently every $a + b$ steps, eventually we reach b if we haven't already reached $-a$. The expected time is bounded by $(a + b)/p^{a+b}$.

end of the $(i + 1)$-th interval. Since this is just one standard deviation, it occurs with constant probability, so after a finite expected number of intervals, we will reach $+a$. Since there are infinitely many intervals, we reach $+a$ with probability 1.

We also have that $0 < Z_t \leq \max\Big((q/p)^a, (q/p)^b\Big)$ for all $t \leq \tau$. This gives us bounded range, so the finite-time/bounded-range case of OST applies.

We thus have $\mathrm{E}[Z_\tau] = p_b(q/p)^b + (1-p_b)(q/p)^{-a} = \mathrm{E}[Z_0] = (q/p)^0 = 1$. Solving for p_b gives

$$p_b = \frac{1-(q/p)^{-a}}{(q/p)^b - (q/p)^{-a}}.$$

(Note that this only makes sense if $q \neq p$.)

As a test, if $-a = -1$ and $+b = +1$, then we get

$$\begin{aligned} p_b &= \frac{1-p/q}{q/p - p/q} \\ &= \frac{pq-p^2}{q-p} \\ &= \frac{p(q-p)}{q-p} \\ &= p, \end{aligned}$$

which is what we would expect since we hit $-a$ or $+b$ on the first step.

A more interesting case is if we set $p < 1/2$. Then $q/p > 1$, and

$$\begin{aligned} p_b &= \frac{1-(q/p)^{-a}}{(q/p)^b - (q/p)^{-a}} \\ &= (q/p)^{-b} \cdot \frac{1-(q/p)^{-a}}{1-(q/p)^{-a-b}} \\ &< (q/p)^{-b}. \end{aligned}$$

Any walk that is biased against us will be an exponentially improbable hill to climb.

2. Now suppose we want to know $\mathrm{E}[\tau]$, the average time at which we first hit a or b. We already argued $\mathrm{E}[\tau]$ is finite, and it's easy to see that $\{Y_t\}$ has bounded increments, so we can use the finite-expected-time/bounded-increment case of OST to get $\mathrm{E}[Y_\tau] = \mathrm{E}[Y_0] = 0$, or $\mathrm{E}[X_\tau - (p-q)\tau] = 0$. It follows that $\mathrm{E}[\tau] = \mathrm{E}[X_\tau]/(p-q)$.

   But we can compute $\mathrm{E}[X_\tau]$, since it is just $p_b b - (1-p_b)a$. So $\mathrm{E}[\tau] = \frac{p_b b - (1-p_b)a}{p-q}$. If $p > q$, and a and b are both large enough to make p_b very close to 1, this will be approximately $b/(p-q)$, the time to climb to b using our average return of $p-q$ per step.

9.4.2 Wald's equation

Suppose we run a Las Vegas algorithm until it succeeds, and the i-th attempt costs X_i, where all the X_i are independent, satisfy $0 \le X_i \le c$ for some c, and have a common mean $\mathrm{E}[X_i] = \mu$.

Let N be the number of times we run the algorithm. Since we can tell when we are done, N is a stopping time with respect to some filtration $\{\mathcal{F}_i\}$ to which the X_i are adapted.[6] Suppose also that $\mathrm{E}[N]$ exists. What is $\mathrm{E}\left[\sum_{i=1}^{N} X_i\right]$?

If N were not a stopping time, this might be a very messy problem indeed. But when N is a stopping time, we can apply it to the martingale $Y_t = \sum_{i=1}^{t}(X_i - \mu)$. This has bounded increments ($0 \le X_i \le c$, so $-c \le X_i - \mathrm{E}[X_i] \le c$), and we've already said $\mathrm{E}[N]$ is finite (which implies $\Pr[N < \infty] = 1$), so Theorem 9.3.1 applies. We thus have

$$\begin{aligned} 0 &= \mathrm{E}[Y_N] \\ &= \mathrm{E}\left[\sum_{i=1}^{N}(X_i - \mu)\right] \\ &= \mathrm{E}\left[\sum_{i=1}^{N} X_i\right] - \mathrm{E}\left[\sum_{i=1}^{N} \mu\right] \\ &= \mathrm{E}\left[\sum_{i=1}^{N} X_i\right] - \mathrm{E}[N]\,\mu. \end{aligned}$$

Rearranging this gives **Wald's equation**:

$$\mathrm{E}\left[\sum_{i=1}^{N} X_i\right] = \mathrm{E}[N]\,\mu. \tag{9.4.1}$$

This is the same formula as in §3.4.3.1, but we've eliminated the bound on N and allowed for much more dependence between N and the X_i.[7]

9.4.3 Maximal inequalities

Suppose we have a martingale $\{X_i\}$ with $X_i \ge 0$ always, and we want to bound $\max_{i \le n} X_i$. We can do this using the Optional Stopping Theorem:

[6] A stochastic process $\{X_t\}$ is **adapted** to a filtration $\{\mathcal{F}_t\}$ if each X_t is $\mathcal{F}_t$-measurable.

[7] In fact, looking closely at the proof reveals that we don't even need the X_i to be independent of each other. We just need that $\mathrm{E}[X_{i+1} \mid \mathcal{F}_i] = \mu$ for all i to make $(Y_t, \mathcal{F}_t)$ a martingale. But if we don't carry any information from one iteration of our Las Vegas algorithm to the next, we'll get independence anyway. So the big payoff is not having to worry about whether N has some devious dependence on the X_i.

Lemma 9.4.1. *Let $\{X_i\}$ be a martingale with $X_i \geq 0$. Then for any fixed n,*

$$\Pr\left[\max_{i \leq n} X_i \geq \alpha\right] \leq \frac{\mathrm{E}[X_0]}{\alpha}. \tag{9.4.2}$$

Proof. The idea is to pick a stopping time τ such that $\max_{i \leq n} X_i \geq \alpha$ if and only if $X_\tau \geq \alpha$.

Let τ be the first time such that $X_\tau \geq \alpha$ or $\tau \geq n$. Then τ is a stopping time for $\{X_i\}$, since we can determine from $X_0, \ldots, X_t$ whether $\tau \leq t$ or not. We also have that $\tau \leq n$ always, which is equivalent to $\tau = \min(\tau, n)$. Finally, $X_\tau \geq \alpha$ means that $\max_{i \leq n} X_i \geq X_\tau \geq \alpha$, and conversely if there is some $t \leq n$ with $X_t = \max_{i \leq n} X_i \geq \alpha$, then τ is the first such t, giving $X_\tau \geq \alpha$.

Lemma 9.3.2 says $\mathrm{E}[X_\tau] = \mathrm{E}[X_0]$. So Markov's inequality gives $\Pr[\max_{i \leq n} X_i \geq \alpha] = \Pr[X_\tau \geq \alpha] \leq \frac{\mathrm{E}[X_\tau]}{\alpha} = \frac{\mathrm{E}[X_0]}{\alpha}$, as claimed. □

Lemma 9.4.1 is a special case of **Doob's martingale inequality**, which says that for a non-negative *submartingale* $\{X_i\}$,

$$\Pr\left[\max_{i \leq n} X_i \geq \alpha\right] \leq \frac{\mathrm{E}[X_n]}{\alpha}. \tag{9.4.3}$$

The proof is similar, but requires showing first that $\mathrm{E}[X_\tau] \leq \mathrm{E}[X_n]$ when $\tau \leq n$ is a stopping time and $\{X_i\}$ is a submartingale.

Doob's martingale inequality is what you get if you generalize Markov's inequality to martingales. The analogous generalization of Chebyshev's inequality is **Kolmogorov's inequality**, which says:

Lemma 9.4.2. *For sums $S_i = \sum_{j=1}^{i} X_j$ of independent random variables $X_1, X_2, \ldots, X_n$ with $\mathrm{E}[X_i] = 0$,*

$$\Pr\left[\max_{i \leq n} |S_i| \geq \alpha\right] \leq \frac{\mathrm{Var}[S]}{\alpha^2}. \tag{9.4.4}$$

Proof. Let $Y_i = S_i^2 - \mathrm{Var}[S_i]$. Then $\{Y_i\}$ is a martingale. This implies that $\mathrm{E}[Y_n] = \mathrm{E}[S_n^2] - \mathrm{Var}[S] = Y_0 = 0$ and thus that $\mathrm{E}[S_n^2] = \mathrm{Var}[S]$. It's easy to see that $\{S_i^2\}$ is a submartingale since partial sums can only increase over time. Now apply (9.4.3). □

In general, because we can always stop updating a martingale once we hit a particular threshold, other martingale concentration bounds like Azuma-Hoeffding will also apply to the maximum or minimum of a martingale over a given interval.

9.4.4 Waiting times for patterns

Let's suppose we flip coins until we see some pattern appear: for example, we might flip coins until we see HTHH. What is the expected number of coin-flips until this happens?

A very clever trick due to Li [Li80] solves this problem exactly using the Optional Stopping Theorem. Suppose our pattern is $x_1 x_2 \dots x_k$. We imagine an army of gamblers, one of which shows up before each coin-flip. Each gambler starts by betting \$1 that next coin-flip will be x_1. If they win, they bets \$2 that the next coin-flip will be x_2, continuing to play double-or-nothing until either they lose (and is down \$1) or wins her last bet on x_k (and is up $2^k - 1$). Because each gambler's winnings form a martingale, so does their sum, and so the expected total return of all gamblers up to the stopping time τ at which our pattern first occurs is 0.

We can now use this fact to compute $\mathrm{E}[\tau]$. When we stop at time τ, we have one gambler who has won $2^k - 1$. We may also have other gamblers who are still in play. For each i with $x_1 \dots x_i = x_{k-i+1} \dots x_k$, there will be a gambler with net winnings $\sum_{j=1}^{i} 2^{j-1} = 2^i - 1$. The remaining gamblers will all be at -1.

Let $\chi_i = 1$ if $x_1 \dots x_i = x_{k-i+1} \dots x_k$, and 0 otherwise. Then the number of losers is given by $\tau - \sum_{i=1}^{k} \chi_i$ and the total expected payoff is

$$\begin{aligned}
\mathrm{E}[X_\tau] &= \mathrm{E}\left[-(\tau - \sum_{i=1}^{k} \chi_i) + \sum_{i=1}^{k} \chi_i \left(2^i - 1\right)\right] \\
&= \mathrm{E}\left[-\tau + \sum_{i=1}^{k} \chi_i \left(2^i\right)\right] \\
&= 0.
\end{aligned}$$

It follows that $\mathrm{E}[\tau] = \sum_{i=1}^{k} \chi_i 2^i$.

As a quick test, the pattern H has $\mathrm{E}[\tau] = 2^1 = 2$. This is consistent with what we know about geometric distributions.

For a longer example, the pattern HTHH only overlaps with its prefix H, so in this case we have $\mathrm{E}[\tau] = \sum \chi_i 2^i = 16 + 2 = 18$. But HHHH overlaps with all of its prefixes, giving $\mathrm{E}[\tau] = 16 + 8 + 4 + 2 = 30$. At the other extreme, THHH has no overlap at all and gives $\mathrm{E}[\tau] = 16$.

In general, for a pattern of length k, we expect a waiting time somewhere between 2^k and $2^{k+1} - 2$—almost a factor of 2 difference depending on how much overlap we get.

This analysis generalizes in the obvious way to biased coins and larger alphabets. See the paper [Li80] for details.

Chapter 10

Markov chains

A (discrete time) **Markov chain** is a sequence of random variables $X_0, X_1, X_2, \ldots$, which we think of as the position of some particle at increasing times in $\mathbb{N}$, where the distribution of X_{t+1} depends only on the value of X_t. A typical example of a Markov chain is a random walk on a graph: each X_t is a node in the graph, and a step moves to one of the neighbors of the current node chosen at random, each with equal probability.

Markov chains come up in randomized algorithms both because the execution of any randomized algorithm is, in effect, a Markov chain (the random variables are the states of the algorithm); and because we can often sample from distributions that are difficult to sample from directly by designing a Markov chain that converges to the distribution we want. Algorithms that use this latter technique are known as **Markov chain Monte Carlo** algorithms, and rely on the fundamental fact that a Markov chain that satisfies a few straightforward conditions will always converge in the limit to a **stationary distribution**, no matter what state it starts in.

An example of this technique that predates randomized algorithms is card shuffling: each permutation of the deck is a state, and the shuffling operation sends the deck to a new state each time it is applied. Assuming the shuffling operation is not too deterministic, it is possible to show that enough shuffling will eventually produce a state that is close to being a uniform random permutation. The big algorithmic question for this and similar Markov chains is how quickly this happens: what is the **mixing time** of the Markov chain, a measure of how long we have to run it to get close to its limit distribution (this notion is defined formally in §10.2.3). Many of the techniques in this chapter will be aimed at finding bounds on the mixing time for particular Markov processes.

If you want to learn more about Markov chains than presented here, they are usually covered in general probability textbooks (for example, in [Fel68] or [GS01]), mentioned in many linear algebra textbooks [Str03], covered in some detail in stochastic processes textbooks [KT75], and covered in exquisite detail in many books dedicated specifically to the subject [KS76, KSK76]. Good sources for mixing times for Markov chains are the textbook of Levin, Peres, and Wilmer [LPW09] and the survey paper by Montenegro and Tetali [MT05]. An early reference on the mixing times for random walks on graphs that helped inspire much subsequent work is the Aldous-Fill manuscript [AF01], which can be found on-line at `http://www.stat.berkeley.edu/~aldous/RWG/book.html`.

10.1 Basic definitions and properties

A **Markov chain** or **Markov process** is a stochastic process[1] where the distribution of X_{t+1} depends only on the value of X_t and not any previous history. Formally, this means that

$$\Pr[X_{t+1} = j \mid X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0] = \Pr[X_{t+1} = j \mid X_t = i_t]. \tag{10.1.1}$$

A stochastic process with this property is called **memoryless**: at any time, you know where you are, and you can figure out where you are going, but you don't know where you were before.

The **state space** of the chain is just the set of all values that each X_t can have. A Markov chain is **finite** or **countable** if it has a finite or countable state space, respectively. We'll mostly be interested in finite Markov chains (since we have to be able to fit them inside our computer), but countable Markov chains will come up in some contexts.[2]

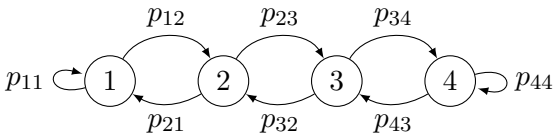
We'll also assume that our Markov chains are **homogeneous**, which means that $\Pr[X_{t+1} = j \mid X_t = i]$ doesn't depend on t.

[1] A **stochastic process** is just a sequence of random variables $\{S_t\}$, where we usually think of t as representing time and the sequence as representing the evolution of some system over time. Here we are considering discrete-time processes, where t will typically be a non-negative integer.

[2] If the state space is not countable, we run into the same measure-theoretic issues as with continuous random variables, and have to replace (10.1.1) with the more general condition that

$$\mathrm{E}\left[1_{[X_t \in A]} \;\middle|\; X_t, X_{t-1}, \ldots, X_0\right] = \mathrm{E}\left[1_{[X_t \in A]} \;\middle|\; X_t\right],$$

provided A is measurable with respect to some appropriate σ-algebra. We don't really want to deal with this, and for the most part we don't have to, so we won't.



$$\begin{bmatrix} p_{11} & p_{12} & 0 & 0 \\ p_{21} & 0 & p_{23} & 0 \\ 0 & p_{32} & 0 & p_{34} \\ 0 & 0 & p_{43} & p_{44} \end{bmatrix}$$

Figure 10.1: Drawing a Markov chain as a directed graph. Nodes represent states. Edges, labeled with probabilities, represent possible transitions. Zero-probability transitions are usually omitted. The corresponding transition matrix is shown on the right.

For a homogeneous countable Markov chain, we can describe its behavior completely by giving the state space and the one-step **transition probabilities** $p_{ij} = \Pr[X_{t+1} = j \mid X_t = i]$. Given p_{ij}, we can calculate two-step transition probabilities

$$\begin{aligned} p_{ij}^{(2)} &= \Pr[X_{t+2} = j \mid X_t = i] \\ &= \sum_k \Pr[X_{t+2} = j \mid X_{t+1} = k] \Pr[X_{t+1} = k \mid X_t = i] \\ &= \sum_k p_{ik} p_{kj}. \end{aligned}$$

This is identical to the formula for matrix multiplication. For a Markov chain with n states, we can specify the transition probabilities p_{ij} using an $n \times n$ **transition matrix** P with $P_{ij} = p_{ij}$, and the two-step transition probabilities are given by $p^{(2)})_{ij} = P_{ij}^2$. More generally, the t-step transition probabilities are given by $p_{ij}^{(t)} = (P^t)_{ij}$.

Conversely, given any matrix with non-negative entries where the rows sum to 1 ($\sum_j P_{ij} = 1$, or $P\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ in the second equation stands for the all-ones vector), there is a corresponding Markov chain given by $p_{ij} = P_{ij}$. Such a matrix is called a **stochastic matrix**; and for every stochastic matrix there is a corresponding finite Markov chain and vice versa.

The general formula for $(s+t)$-step transition probabilities is that $p_{ij}^{(s+t)} = \sum_k p_{ik}^{(s)} p_{kj}^{(t)}$. This is known as the **Chapman-Kolmogorov equation** and is equivalent to the matrix identity $P^{s+t} = P^s P^t$. It is also similar to the formula for counting paths in a directed graph, and we can use the correspondence between matrices and (labeled) directed graphs to give visual depictions of particular Markov chains, as shown in Figure 10.1.

A distribution over states of a finite Markov chain at some time t can be given by a row vector x, where $x_i = Pr[X_t = i]$. To compute the

distribution at time $t+1$, we use the law of total probability: $\Pr[X_{t+1} = j] = \sum_i \Pr[X_t = i] \Pr[X_{t+1} = j \mid X_t = i] = \sum_i x_i p_{ij}$. Again we have the formula for matrix multiplication (where we treat x as a $1 \times n$ matrix); so the distribution vector at time $t+1$ is just xP, and at time $t+n$ is xP^n.

We like Markov chains for two reasons:

1. They describe what happens in a randomized algorithm; the state space is just the set of all states of the algorithm, and the Markov property holds because the algorithm can't remember anything that isn't part of its state. So if we want to analyze randomized algorithms, we will need to get good at analyzing Markov chains.

2. They can be used to do sampling over interesting distributions. Under appropriate conditions (see below), the state of a Markov chain converges to a **stationary distribution**. If we build the right Markov chain, we can control what this stationary distribution looks like, run the chain for a while, and get a sample close to the stationary distribution.

In both cases we want to have a bound on how long it takes the Markov chain to converge, either because it tells us when our algorithm terminates, or because it tells us how long to mix it up before looking at the current state.

10.1.1 Examples

- A fair ± 1 random walk. The state space is $\mathbb{Z}$, the transition probabilities are $p_{ij} = 1/2$ if $|i - j| = 1$, 0 otherwise. This is an example of a Markov chain that is also a martingale.

- A fair ± 1 random walk on a cycle. As above, but now the state space is $\mathbb{Z}/m$, the integers mod m. This is a finite Markov chain. It is also in some sense a martingale, although we usually don't define martingales over finite groups.

- Random walks with absorbing and/or reflecting barriers.

- Random walk on a graph $G = (V, E)$. The state space is V, the transition probabilities are $p_{uv} = 1/d(u)$ if $uv \in E$.

  One can also have more general transition probabilities, where the probability of traversing a particular edge is a property of the edge and not the degree of its source. In principle we can represent any Markov

chain as a random walk on graph in this way: the states become vertices, and the transitions become edges, each labeled with its transition probability. It's conventional in this representation to exclude edges with probability 0 and include self-loops for any transitions $i \to i$.

If the resulting graph is small enough or has a nice structure, this can be a convenient way to draw a Markov chain.

- The Markov chain given by $X_{t+1} = X_t + 1$ with probability $1/2$, and 0 with probability $1/2$. The state space is $\mathbb{N}$.
- A finite-state machine running on a random input. The sequence of states acts as a Markov chain, assuming each input symbol is independent of the rest.
- A classic randomized algorithm for 2-SAT, due to Papadimitriou [Pap91]. Each state is a truth-assignment. The transitional probabilities are messy but arise from the following process: pick an unsatisfied clause, pick one of its two variables uniformly at random, and invert it. Then there is an absorbing state at any satisfying assignment. With a bit of work, it can be shown that the Hamming distance between the current assignment and some satisfying assignment follows a random walk that is either unbiased or biased toward 0, giving a satisfying assignment after $O(n^2)$ steps on average.[3] This algorithm is not necessarily all that good, because there is a clever deterministic algorithm that solves 2-SAT in time linear in the size of the formula [APT79], but it has the nice property of not being so clever.
- A similar process works for 2-colorability, 3-SAT, 3-colorability, etc., although for **NP**-hard problems, it may take a while to reach an absorbing state. The constructive Lovász Local Lemma proof from §13.3.5 also follows this pattern.

[3] The proof of this is not too hard: given an unsatisfying assignment x and a satisfying assignment y, any clause that is not satisfied by x includes at least one variable that is satisfied by y. If we get lucky and flip this variable, we reduce the distance by one. If not, we increase it by at most 1 (depending on whether y satisfies only one or both variables). So we get a process bounded by an unbiased random walk with a reflecting barrier at n and an absorbing barrier at 0, assuming we don't hit some other satisfying assignment y' first.

10.2 Convergence of Markov chains

We want to use Markov chains for sampling. Typically this means that we have some subset S of the state space, and we want to know what proportion of the states are in S. If we can't sample states from from the state space a priori, we may be able to get a good approximation by running the Markov chain for a while and hoping that it converges to something predictable.

To show this, we will proceed though several steps:

1. We will define a class of distributions, known as **stationary distributions**, that we hope to converge to (§10.2.1

2. We will define a distance between distributions, the **total variation distance** (§10.2.2).

3. We will define the **mixing time** of a Markov chain as the minimum time for the distribution of the position of a particle to get within a given total variation distance of the stationary distribution (§10.2.3).

4. We will describe a technique called **coupling** that can be used to bound total variation distance (§10.2.4), in terms of the probability that two dependent variables X and Y with the distributions we are looking at are or are not equal to each other.

5. We will define **reducible** and **periodic** Markov chains, which have structural properties that prevent convergence to a unique stationary distribution (§10.2.5).

6. We will use a coupling between two copies of a Markov chain to show that *any* Markov chain that does not have these properties does converge in total variation distance to a unique stationary distribution (§10.2.6).

7. Finally, we will show that if we can construct a coupling between two copies of a particular chain that causes both copies to reach the same state quickly on average, then we can use this expected **coupling time** to bound the mixing time of the chain that we defined previously. This will give us a practical tool for showing that many Markov chains not only converge eventually but converge at a predictable rate, so we can tell when it is safe to stop running the chain and take a sample (§10.4).

Much of this section follows the approach of Chapter 4 of Levin *et al.* [LPW09].

10.2.1 Stationary distributions

For a finite Markov chain, there is a transition matrix P in which each row sums to 1. We can write this fact compactly as $P\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is the all-ones column vector. This means that $\mathbf{1}$ is a right eigenvector of P with eigenvalue 1.[4] Because the left eigenvalues of a matrix are equal to the right eigenvalues, this means that there will be at least one left eigenvector π such that $\pi P = \pi$, and in fact it is possible to show that there is at least one such π that represents a probability distribution in that each $\pi_i \geq 0$ and $\sum \pi_i = 1$. Such a distribution is called a **stationary distribution**[5] of the Markov chain, and if π is unique, the probability π_i is called the **stationary probability** for i.

Every finite Markov chain has at least one stationary distribution, but it may not be unique. For example, if the transition matrix is the identity matrix (meaning that the particle never moves), then all distributions are stationary.

If a Markov chain does have a unique stationary distribution, we can calculate it from the transition matrix, by observing that the equations

$$\pi P = P$$

and

$$\pi \mathbf{1} = 1$$

together give $n + 1$ equations in n unknowns, which we can solve for π. (We need an extra equation because the stochastic property of P means that it has rank at most $n - 1$.)

Often this will be impractical, especially if our state space is large enough or messy enough that we can't write down the entire matrix. In these cases we may be able to take advantage of a special property of some Markov chains, called **reversibility**. We'll discuss this in §10.3. For the moment we will content ourselves with showing that we do in fact converge to some unique stationary distribution if our Markov chain has the right properties.

10.2.2 Total variation distance

Definition 10.2.1. *Let X and Y be random variables defined on the same probability space. Then the **total variation distance** between X and Y,*

[4] Given a square matrix A, a vector x is a right eigenvector of A with eigenvalue λ is $Ax = \lambda x$. Similarly, a vector y is a left eigenvalue of A with eigenvalue λ if $yA = \lambda A$.

[5] Spelling is important here. A *stationery distribution* would involve handing out office supplies.

written $d_{TV}(X, Y)$ *is given by*

$$d_{TV}(X, Y) = \sup_A \left(\Pr[X \in A] - \Pr[Y \in A] \right), \tag{10.2.1}$$

where the supremum is taken over all sets A *for which* $\Pr[X \in A]$ *and* $\Pr[Y \in A]$ *are both defined.*[6]

An equivalent definition is

$$d_{TV}(X, Y) = \sup_A |\Pr[X \in A] - \Pr[Y \in A]|.$$

The reason this is equivalent is that if $\Pr[X \in A] - \Pr[Y \in A]$ is negative, we can replace A by its complement.

Less formally, given any test set A, X and Y show up in A with probabilities that differ by at most $d_{TV}(X, Y)$. This is usually what we want for sampling, since this says that if we are testing some property (represented by A) of the states we are sampling, the answer ($\Pr[X \in A]$) that we get for how likely this property is to occur is close to the correct answer ($\Pr[Y \in A]$).

Total variation distance is a property of distributions, and is not affected by dependence between X and Y. For finite Markov chains, we can define the total variation distance between two distributions x and y as

$$d_{TV}(x, y) = \max_A \sum_{i \in A} (x_i - y_i) = \max_A \left| \sum_{i \in A} (x_i - y_i) \right|.$$

A useful fact is that $d_{TV}(x, y)$ is directly connected to the ℓ_1 distance between x and y. If we let $B = \{i \mid x_i \geq y_i\}$, then

$$\begin{aligned} d_{TV}(x, y) &= \max_A \sum_{i \in A} (x_i - y_i) \\ &\leq \sum_{i \in B} (x_i - y_i), \end{aligned}$$

because if A leaves out an element of B, or includes and element of $\overline{B}$, this can only reduce the sum. But if we consider $\overline{B}$ instead, we get

$$\begin{aligned} d_{TV}(y, x) &= \max_A \sum_{i \in A} (y_i - x_i) \\ &\leq \sum_{i \in B} (y_i - x_i). \end{aligned}$$

[6] For discrete random variables, this just means all A, since we can write $\Pr[X \in A]$ as $\sum_{x \in A} \Pr[X = x]$; we can also replace sup with max for this case. For continuous random variables, we want that $X^{-1}(A)$ and $Y^{-1}(A)$ are both measurable. If our X and Y range over the states of a countable Markov chain, we will be working with discrete random variables, so we can just consider all A.

Now observe that

$$\begin{aligned}\|x-y\|_1 &= \sum_i |x_i - y_i| \\ &= \sum_{i\in B}(x_i - y_i) + \sum_{i\in\overline{B}}(y_i - x_i) \\ &= d_{TV}(x,y) + d_{TV}(y,x) \\ &= 2d_{TV}(x,y).\end{aligned}$$

So $d_{TV}(x,y) = \frac{1}{2}\|x-y\|_1$.

10.2.2.1 Total variation distance and expectation

Sometimes it's useful to translate a bound on total variation to a bound on the error when getting the expectation of a random variable. The following lemma may be handy:

Lemma 10.2.2. *Let x and y be two distributions of some discrete random variable Z. Let $\mathrm{E}_x(Z)$ and $\mathrm{E}_y(Z)$ be the expectations of Z with respect to each of these distributions. Suppose that $|Z| \le M$ always. Then*

$$|\mathrm{E}_x(Z) - \mathrm{E}_y(Z)| \le 2M \cdot d_{TV}(x,y). \tag{10.2.2}$$

Proof. Compute

$$\begin{aligned}|\mathrm{E}_x(Z) - \mathrm{E}_y(Z)| &= \left|\sum_z z\left(\Pr_x(Z=z) - \Pr_y(Z=z)\right)\right| \\ &\le \sum_z |z| \cdot \left|\Pr_x(Z=z) - \Pr_y(Z=z)\right| \\ &\le M \sum_z \left|\Pr_x(Z=z) - \Pr_y(Z=z)\right| \\ &\le M\,\|x-y\|_1 \\ &= 2M \cdot d_{TV}(x,y).\end{aligned}$$

□

10.2.3 Mixing time

We are going to show that well-behaved finite Markov chains eventually converge to some stationary distribution π in total variation distance. This

means that for any $\epsilon > 0$, there is a **mixing time** $t_{\text{mix}}(\epsilon)$ such that for any initial distribution x and any $t \geq t_{\text{mix}}(\epsilon)$,

$$d_{TV}(xP^t, \pi) \leq \epsilon.$$

It is common to standardize ϵ as $1/4$: if we write just t_{mix}, this means $t_{\text{mix}}(1/4)$. The choice of $1/4$ is somewhat arbitrary, but has some nice technical properties that we will see below.

10.2.4 Coupling of Markov chains

A **coupling** of two random variables X and Y is a joint distribution on $\langle X, Y \rangle$ that gives the correct marginal distribution for each of X and Y while creating a dependence between them with some desirable property (for example, minimizing total variation distance or maximize $\Pr[X = Y]$).

We will use couplings between Markov chains to prove convergence. Here we take two copies of the chain, once of which starts in an arbitrary distribution, and one of which starts in the stationary distribution, and show that we can force them to converge to each other by carefully correlating their transitions. Since the second chain is always in the stationary distribution, this will show that the first chain converges to the stationary distribution as well.

The tool that makes this work is the **Coupling Lemma**:[7]

Lemma 10.2.3. *For any discrete random variables X and Y,*

$$d_{TV}(X, Y) \leq \Pr[X \neq Y].$$

[7] It turns out that the bound in the Coupling Lemma is tight in the following sense: for any given distributions on X and Y, there exists a joint distribution giving these distributions such that $d_{TV}(X, Y)$ is exactly equal to $\Pr[X \neq Y]$ when X and Y are sampled from the joint distribution. For discrete distributions, the easiest way to construct the joint distribution is first to let to let $Y = X = i$ for each i with probability $\min(\Pr[X = i], \Pr[Y = i])$, and then distribute the remaining probability for X over all the cases where $\Pr[X = i] > \Pr[Y = i]$ and similarly for Y over all the cases where $\Pr[Y = i] > \Pr[X = i]$. Looking at the unmatched values for X gives $\Pr[X \neq Y] \leq \sum_{\{x \mid \Pr X = i > \Pr Y = i\}} (\Pr[X = i] - \Pr[Y = i]) \leq d_{TV}(X, Y)$. So in this case $\Pr[X \neq Y] = d_{TV}(X, Y)$.

Unfortunately, the fact that there always exists a perfect coupling in this sense does not mean that we can express it in any convenient way, or that even if we could, it would arise from the kind of causal, step-by-step construction that we will use for couplings between Markov processes.

Proof. Let A be any set for which $\Pr[X \in A]$ and $\Pr[Y \in A]$ are defined. Then

$$\Pr[X \in A] = \Pr[X \in A \wedge Y \in A] + \Pr[X \in A \wedge Y \notin A],$$
$$\Pr[Y \in A] = \Pr[X \in A \wedge Y \in A] + \Pr[X \notin A \wedge Y \in A],$$

and thus

$$\begin{aligned}\Pr[X \in A] - \Pr[Y \in A] &= \Pr[X \in A \wedge Y \notin A] - \Pr[X \notin A \wedge Y \in A] \\ &\leq \Pr[X \in A \wedge Y \notin A] \\ &\leq \Pr[X \neq Y].\end{aligned}$$

Since this holds for any particular set A, it also holds when we take the maximum over all A to get $d_{TV}(X, Y)$. □

For Markov chains, our goal will be to find a useful coupling between a sequence of random variables $X_0, X_1, X_2, \ldots$ corresponding to the Markov chain starting in an arbitrary distribution with a second sequence $Y_0, Y_1, Y_2, \ldots$ corresponding to the same chain starting in a stationary distribution. What will make a coupling useful is if $\Pr[X_t \neq Y_t]$ is small for reasonably large t: since Y_t has the stationary distribution, this will show that $d_{TV}(xP^t, \pi)$ is also small.

Our first use of this technique will be to show, using a rather generic coupling, that Markov chains with certain nice properties converge to their stationary distribution in the limit. Later we will construct specialized couplings for particular Markov chains to show that they converge quickly. But first we will consider what properties a Markov chain must have to converge at all.

10.2.5 Irreducible and aperiodic chains

Not all chains are guaranteed to converge to their stationary distribution. If some states are not reachable from other states, it may be that starting in one part of the chain will keep us from ever reaching another part of the chain. Even if the chain is not disconnected in this way, we might still not converge if the distribution oscillates back and forth due to some periodicity in the structure of the chain. But if these conditions do not occur, then we will be able to show convergence.

Let p_{ij}^t be $\Pr[X_t = j \mid X_0 = i]$.

A Markov chain is **irreducible** if, for all states i and j, there exists some t such that $p_{ij}^t \neq 0$. This says that we can reach any state from any other

state if we wait long enough. If we think of a directed graph of the Markov chain where the states are vertices and each edge represents a transition that occurs with nonzero probability, the Markov chain is irreducible if its graph is strongly connected.

The **period** of a state i of a Markov chain is $\gcd(\{t > 0 \mid p_{ii}^t \neq 0\})$. If the period of i is m, then starting from i we can only return to i at times that are multiples of m. If $m = 1$, state i is said to be **aperiodic**. A Markov chain as a whole is aperiodic if all of its states are aperiodic. In graph-theoretic terms, this means that the graph of the chain is not k-partite for any $k > 1$. Reversible chains are also an interesting special case: if a chain is reversible, it can't have a period greater than 2, since we can always step off a node and step back.

If our Markov chain is not aperiodic, we can make it aperiodic by flipping a coin at each step to decide whether to move or not. This gives a **lazy Markov chain** whose transition probabilities are given by $\frac{1}{2}p_{ij}$ when $i \neq j$ and $\frac{1}{2} + \frac{1}{2}p_{ij}$ when $i = j$. This doesn't affect the stationary distribution: if we replace our transition matrix P with a new transition matrix $\frac{P+I}{2}$, and $\pi P = \pi$, then $\pi\left(\frac{P+I}{2}\right) = \frac{1}{2}\pi P + \frac{1}{2}\pi I = \frac{1}{2}\pi + \frac{1}{2}\pi = \pi$.

Unfortunately there is no quick fix for reducible Markov chains. But since we will often be designing the Markov chains we will be working with, we can just take care to make sure they are not reducible.

We will later need the following lemma about aperiodic Markov chains, which is related to the **Frobenius problem** of finding the minimum value that cannot be constructed using coins of given denominations:

Lemma 10.2.4. *Let i be an aperiodic state of some Markov chain. Then there is a time t_0 such that $p_{ii}^t \neq 0$ for all $t \geq t_0$.*

Proof. Let $S = \{t \mid p_{ii}(t) \neq 0\}$. Since $\gcd(S) = 1$, there is a finite subset S' of S such that $\gcd S' = 1$. Write the elements of S' as $m_1, m_2, \ldots, m_k$ and let $M = \prod_{j=1}^k m_j$. From the extended Euclidean algorithm, there exist integer coefficients a_j with $|a_j| \leq M/m_j$ such that $\sum_{j=1}^k a_j m_j = 1$. We would like to use each a_j as the number of times to go around the length m_j loop from i to i. Unfortunately many of these a_j will be negative.

To solve this problem, we replace a_j with $b_j = a_j + M/m_j$. This makes all the coefficients non-negative, and gives $\sum_{j=1}^k b_j m_j = kM + 1$. This implies that there is a sequence of loops that gets us from i back to i in $kM + 1$ steps, or in other words that $p_{ii}^{kM+1} \neq 0$. By repeating this sequence ℓ times, we can similarly show that $p_{ii}^{\ell kM+\ell} \neq 0$ for any ℓ.

We can also pad any of these sequences out by as many copies of M as

we like. In particular, given $\ell kM + \ell$, where $\ell \in \{0, \ldots, M-1\}$, we can add $(kM - \ell)M$ to it to get $(kM)^2 + \ell$. This means that we can express any $t \in \{(kM)^2, \ldots, (kM)^2 + M - 1\}$ as a sum of elements of S, or equivalently that $p_{ii}^t \neq 0$ for any such t. But for larger t, we can just add in more copies of M. So in fact $p_{ii}^t \neq 0$ for all $t \geq t0 = (kM)^2$. □

10.2.6 Convergence of finite irreducible aperiodic Markov chains

We can now show:

Theorem 10.2.5. *Any finite irreducible aperiodic Markov chain converges to a unique stationary distribution in the limit.*

Proof. Consider two copies of the chain $\{X_t\}$ and $\{Y_t\}$, where X_0 starts in some arbitrary distribution x and Y_0 starts in a stationary distribution π. Define a coupling between $\{X_t\}$ and $\{Y_t\}$ by the rule: (a) if $X_t \neq Y_t$, then $\Pr[X_{t+1} = j \wedge Y_{t+1} = j' \mid X_t = i \wedge Y_t = i'] = p_{ij}p_{i'j'}$; and (b) if $X_t = Y_t$, then $\Pr[X_{t+1} = Y_{t+1} = j \mid X_t = Y_t = i] = p_{ij}$. Intuitively, we let both chains run independently until they collide, after which we run them together. Since each chain individually moves from state i to state j with probability p_{ij} in either case, we have that X_t evolves normally and Y_t remains in the stationary distribution.

Now let us show that $d_{TV}(xP^t, \pi) \leq \Pr[X_t \neq Y_t]$ goes to zero in the limit. Pick some state i. Let r be the maximum over all states j of the **first passage time** f_{ji} where f_{ji} is the minimum time t such that $p_{ji}^t \neq 0$. Let s be a time such that $p_{ii}^t \neq 0$ for all $t \geq s$ (the existence of such an s is given by Lemma 10.2.4).

Suppose that at time $\ell(r+s)$, where $\ell \in \mathbb{N}$, $X_{\ell(r+s)} = j \neq j' = Y_{\ell(r+s)}$. Then there are times $\ell(r+s)+u$ and $\ell(r+s)+u'$, where $u, u' \leq r$, such that X reaches i at time $\ell(r+s)+u$ and Y reaches i at time $\ell(r+s)+u'$ with nonzero probability. Since $(r+s-u) \leq s$, then having reached i at these times, X and Y both return to i at time $\ell(r+s)+(r+s) = (\ell+1)(r+s)$ with nonzero probability. Let $\epsilon > 0$ be the product of these nonzero probabilities; then $\Pr\left[X_{(\ell+1)(r+s)} \neq Y_{(\ell+1)(r+s)}\right] \leq (1-\epsilon)\Pr\left[X_{\ell(r+s)} = Y_{\ell(r+s)}\right]$, and in general we have $\Pr[X_t \neq Y_t] \leq (1-\epsilon)^{\lfloor t/(r+s) \rfloor}$, which goes to zero in the limit. This implies that $d_{TV}(xP^t, \pi)$ also goes to zero in the limit (using the Coupling Lemma), and since any initial distribution (including a stationary distribution) converges to π, π is the unique stationary distribution as claimed. □

This argument requires that the chain be finite, because otherwise we cannot take the maximum over all the first passage times. For infinite Markov chains, it is not always enough to be irreducible and aperiodic to converge to a stationary distribution (or even to have a stationary distribution at all). However, with some additional conditions a similar result can be shown: see for example [GS01, §6.4].

10.3 Reversible chains

A Markov chain with transition probabilities p_{ij} is **reversible** if there is a distribution π such that, for all i and j,

$$\pi_i p_{ij} = \pi_j p_{ji}. \tag{10.3.1}$$

These are called the **detailed balance equations**—they say that in the stationary distribution, the probability of seeing a transition from i to j is equal to the probability of seeing a transition from j to i). If this is the case, then $\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j$, which means that π is stationary.

It's worth noting that this works for countable chains even if they are not finite, because the sums always converge since each term is non-negative and $\sum_i \pi_i p_{ij}$ is dominated by $\sum_i \pi_i = 1$. However, it may not be the case for any particular p that there exists a corresponding stationary distribution π. If this happens, the chain is not reversible.

10.3.1 Stationary distributions

The detailed balance equations often give a very quick way to compute the stationary distribution, since if we know π_i, and $p_{ij} \neq 0$, then $\pi_j = \pi_i p_{ij}/p_{ji}$. If the transition probabilities are reasonably well-behaved (for example, if $p_{ij} = p_{ij}$ for all i, j), we may even be able to characterize the stationary distribution up to a constant multiple even if we have no way to efficiently enumerate all the states of the process.

What (10.3.1) says is that if we start in the stationary distribution and observe either a forward transition $\langle X_0, X_1\rangle$ or a backward transition $\langle X_1, X_0\rangle$, we can't tell which is which; $\Pr[X_0 = i \wedge X_1 = j] = \Pr[X_1 = i \wedge X_0 = j]$. This extends to longer sequences. The probability that $X_0 = i$, $X_1 = j$, and $X_2 = k$ is given by $\pi_i p_{ij} p_{jk} = p_{ji}\pi_j p_{jk} = p_{ji} p_{kj} \pi_k$, which is the probability that $X_0 = k$, $X_1 = j$, and $X_0 = i$. (A similar argument works for finite sequences of any length.) So a reversible Markov chain is one with no arrow of time in the stationary distribution.

A typical reversible Markov chain is a random walk on a graph, where a step starting from a vertex u goes to one of its neighbors v, which each neighbor chosen with probability $\frac{1}{d(u)}$. This has a stationary distribution

$$\begin{aligned}\pi_u &= \frac{d(u)}{\sum_u d(u)} \\ &= \frac{d(u)}{2|E|},\end{aligned}$$

which satisfies

$$\begin{aligned}\pi_u p_{uv} &= \frac{d(u}{2|E|} \cdot \frac{1}{d(u)} \\ &= \frac{d(v}{2|E|} \cdot \frac{1}{d(v)} \\ &= \pi_v p_{vu}.\end{aligned}$$

If we don't know π in advance, we can often guess it by observing that

$$\pi_i p_{ij} = \pi_j p_{ji}$$

implies

$$\pi_j = \pi_i \frac{p_{ij}}{p_{ji}}, \tag{10.3.2}$$

provided $p_{ij} \neq 0$. This gives us the ability to calculate π_k starting from any initial state i as long as there is some chain of transitions $i = i_0 \to i_1 \to i_2 \to \ldots i_\ell = k$ where each step $i_m \to i_{m+1}$ has $p_{i_m, i_{m+1}} \neq 0$. For a random walk on a graph, this implies that π is unique as long as the graph is connected. This of course only works for reversible chains; if we try to do this with a non-reversible chain, we are likely to get a contradiction.

For example, if we consider a biased random walk on the n-cycle, which moves from i to $(i+1) \bmod n$ with probability p and in the other direction with probability $q = 1 - p$, then applying (10.3.2) repeatedly would give $\pi_i = \pi_0 \left(\frac{p}{q}\right)^i$. This is not a problem when $p = q = 1/2$, since we get $\pi_i = \pi_0$ for all i and can deduce that $\pi_i = 1/n$ is the unique stationary distribution. But if we try it for $p = 2/3$, then we get $\pi_i = \pi_0 2^i$, which is fine up until we hit $\pi_0 = \pi_n = \pi_0 2^n$. So for $p \neq q$, this process is not reversible, which is not surprising if we realize that the $n = 60, p = 1$ case describes precisely the movement of the second hand on a clock.[8]

[8] It happens to be the case that $\pi_i = 1/n$ is a stationary distribution for any value of p, we just can't prove this using (10.3.1).

Reversible chains have a special role in Markov chain theory, because some techniques for proving convergence only apply to reversible chains (see §10.6). They are also handy for sampling from large, irregular state spaces, because by tuning the transition probabilities locally we can often adjust the relative likelihoods of states in the stationary distribution to be closer to what we want (see §10.3.4).

10.3.2 Examples

Random walk on a weighted graph Here each edge has a weight w_{uv} where $0 < w_{uv} = w_{vu} < \infty$, with self-loops permitted. A step of the random walk goes from u to v with probability $w_{uv} / \sum_{v'} w_{uv'}$. It is easy to show that this random walk has stationary distribution $\pi_u = \sum_u w_{uv} / \sum_u \sum_v w_{uv}$, generalizing the previous case, and that the resulting Markov chain satisfies the detailed balance equations.

Random walk with uniform stationary distribution Now let Δ be the maximum degree of the graph, and traverse each each with probability $1/\Delta$, staying put on each vertex u with probability $1 - d(u)/\Delta$. The stationary distribution is uniform, since for each pair of vertices u and v we have $p_{uv} = p_{vu} = 1/\Delta$ if u and v are adjacent and 0 otherwise. This is a special case of the Metropolis-Hastings algorithm (see §10.3.4).

10.3.3 Time-reversed chains

Another way to get a reversible chain is to take an arbitrary chain with a stationary distribution and rearrange it so that it can run both forwards and backwards in time. This is not necessarily useful, but it illustrates the connection between reversible and irreversible chains.

Given a finite Markov chain with transition matrix P and stationary distribution π, define the corresponding **time-reversed chain** with matrix P^* where $\pi_i p_{ij} = \pi_j p^*_{ji}$.

To make sure that this actually works, we need to verify that:

1. The matrix P^* is stochastic:

$$\begin{aligned}\sum_j p^*_{ij} &= \sum_j p_{ji}\pi_j/\pi_i \\ &= \pi_i/\pi_i \\ &= 1.\end{aligned}$$

2. The reversed chain has the same stationary distribution as the original chain:

$$\begin{aligned}\sum_j \pi_j p^*_{ji} &= \sum_j \pi_i p_{ij} \\ &= \pi_i.\end{aligned}$$

3. And that in general P^*'s paths starting from the stationary distribution are a reverse of P's paths starting from the same distribution. For length-1 paths, this is just $\pi_j p^*_{ji} = \pi_i p_{ij}$. For longer paths, this follows from an argument similar to that for reversible chains.

This gives an alternate definition of a reversible chain as a chain for which $P = P^*$.

We can also use time-reversal to generate reversible chains from arbitrary chains. The chain with transition matrix $(P + P^*)/2$ (corresponding to moving 1 step forward or back with equal probability at each step) is always a reversible chain.

Examples:

- Given a biased random walk on a cycle that moves right with probability p and left with probability q, its time-reversal is the walk that moves left with probability p and right with probability q. (Here the fact that the stationary distribution is uniform makes things simple.) The average of this chain with its time-reversal is an unbiased random walk.

- Given the random walk defined by $X_{t+1} = X_t + 1$ with probability $1/2$ and 0 with probability $1/2$, we have $\pi_i = 2^{-i-1}$. This is not reversible (there is a transition from 1 to 2 but none from 2 to 1), but we can reverse it by setting $p^*_{ij} = 1$ for $i = j + 1$ and $p^*_{0i} = 2^{-i-1}$. (Check: $\pi_i p_{ii+1} = 2^{-i-1}(1/2) = \pi_{i+1} p^*_{i+1i} = 2^{-i-2}(1); \pi_i p_{i0} = 2^{-i-1}(1/2) = \pi_0 p^*_{0i} = (1/2)2^{-i-1}$.)

These examples work because the original chains are simple and have clean stationary distributions. Reversed versions of chains with messier stationary distributions are usually messier. In practice, building reversible chains using time-reversal is often painful precisely because we don't have a good characterization of the stationary distribution of the original non-reversible chain. So we will often design our chains to be reversible from the start rather than relying on after-the-fact flipping.

10.3.4 Adjusting stationary distributions with the Metropolis-Hastings algorithm

Sometimes we have a reversible Markov chain, but it doesn't have the stationary distribution we want. The **Metropolis-Hastings algorithm**[MRR+53, Has70] (sometimes just called **Metropolis**) lets us start with a reversible Markov chain P with a known stationary distribution π and convert it to a chain Q on the same states with a different stationary distribution μ, where $\mu_i = f(i)/\sum_j f(j)$ is proportional to some function $f \geq 0$ on states that we can compute easily.

A typical application is that we want to sample according to $\Pr[i \mid A]$, but A is highly improbable (so we can't just use **rejection sampling**, where we sample random points from the original distribution until we find one for which A holds), and $\Pr[i \mid A]$ is easy to compute for any fixed i but tricky to compute for arbitrary events (so we can't use divide-and-conquer). If we let $f(i) \propto \Pr[i \mid A]$, then Metropolis-Hastings will do exactly what we want, assuming it converges in a reasonable amount of time.

Let q be the transition probability for Q. Define, for $i \neq j$,

$$\begin{aligned} q_{ij} &= p_{ij} \min\left(1, \frac{\pi_i f(j)}{\pi_j f(i)}\right) \\ &= p_{ij} \min\left(1, \frac{\pi_i \mu_j}{\pi_j \mu_i}\right) \end{aligned}$$

and let q_{ii} be whatever probability is left over. Now consider two states i and j, and suppose that $\pi_i f(j) \geq \pi_j f(i)$. Then

$$q_{ij} = p_{ij}$$

which gives

$$\mu_i q_{ij} = \mu_i p_{ij},$$

while

$$\begin{aligned} \mu_j q_{ji} &= \mu_j p_{ji} (\pi_j \mu_i / \pi_i \mu_j) \\ &= p_{ji} (\pi_j \mu_i / \pi_i) \\ &= \mu_i (p_{ji} \pi_j) / \pi_i \\ &= \mu_i (p_{ij} \pi_i) / \pi_i \\ &= \mu_i p_{ij} \end{aligned}$$

(note the use of reversibility of P in the second-to-last step). So we have $\mu_j q_{ji} = \mu_i p_{ij} = \mu_i q_{ij}$ and Q is a reversible Markov chain with stationary distribution μ.

We can simplify this when our underlying chain P has a uniform stationary distribution (for example, when it's the random walk on a graph with maximum degree Δ, where we traverse each edge with probability $1/\Delta$). Then we have $\pi_i = \pi_j$ for all i, j, so the new transition probabilities q_{ij} are just $\frac{1}{\Delta} \min(1, f(j)/f(i))$. Most of our examples of reversible chains will be instances of this case.

10.4 The coupling method

In order to use the stationary distribution of a Markov chain to do sampling, we need to have a bound on the rate of convergence to tell us when it is safe to take a sample. There are two standard techniques for doing this: coupling, where we show that a copy of the process starting in an arbitrary state can be made to converge to a copy starting in the stationary distribution; and spectral methods, where we bound the rate of convergence by looking at the second-largest eigenvalue of the transition matrix. We'll start with coupling because it requires less development.

(See also [Gur00] for a survey of the relationship between the various methods.)

Note: these notes will be somewhat sketchy. If you want to read more about coupling, a good place to start might be Chapter 12 of [MU17]; Chapter 4-3 (`http://www.stat.berkeley.edu/~aldous/RWG/Chap4-3.pdf`) of the unpublished but nonetheless famous **Aldous-Fill manuscript** (`http://www.stat.berkeley.edu/~aldous/RWG/book.html`, [AF01]), which is a good place to learn about Markov chains and Markov chain Monte Carlo methods in general; or even an entire book [Lin92]. We'll mostly be using examples from the Aldous-Fill text.

10.4.1 Random walk on a cycle

Let's suppose we do a random walk on $\mathbb{Z}_m$, where to avoid periodicity at each step we stay put with probability $1/2$, move counterclockwise with probability $1/4$, and move clockwise with probability $1/4$: in other words, we are doing a lazy unbiased random walk. What's a good choice for a coupling to show this process converges quickly?

Specifically, we need to create a joint process (X_t, Y_t), where each of the marginal processes X and Y looks like a lazy random walk on $\mathbb{Z}_m$, X_0 has

whatever distribution our real process starts with, and Y_0 has the stationary distribution. Our goal is to structure the combined process so that $X_t = Y_t$ as soon as possible.

Let $Z_t = X_t - Y_t \pmod m$. If $Z_t = 0$, then X_t and Y_t have collided and we will move both together. If $Z_t \neq 0$, then flip a coin to decide whether to move X_t or Y_t; whichever one moves then moves up or down with equal probability. It's not hard to see that this gives a probability of exactly $1/2$ that $X_{t+1} = X_t$, $1/4$ that $X_{t+1} = X_t + 1$, and $1/4$ that $X_{t+1} = X_t - 1$, and similarly for Y_t. So the transition functions for X and Y individually are the same as for the original process.

Whichever way the first flip goes, we get $Z_{t+1} = Z_t \pm 1$ with equal probability. So Z acts as an unbiased random walk on $\mathbb{Z}_m$ with an absorbing barriers at 0; this is equivalent to a random walk on $0 \ldots m$ with absorbing barriers at both endpoints. The expected time for this random walk to reach a barrier starting from an arbitrary initial state is at most $m^2/4$, so if τ is the first time at which $X_\tau = Y_\tau$, we have $\mathrm{E}[\tau] \leq m^2/4$.[9]

Using Markov's inequality, after $t = 2(m^2/4) = m^2/2$ steps we have $\Pr[X_t \neq Y_t] = \Pr[\tau > m^2/2] \leq \frac{\mathrm{E}[\tau]}{m^2/2} \leq 1/2$. We can also iterate the whole argument, starting over in whatever state we are in at time t if we don't converge. This gives at most a $1/2$ chance of not converging for each interval of $m^2/2$ steps. So after $\alpha m^2/2$ steps we will have $\Pr[X_t \neq Y_t] \leq 2^{-\alpha}$. This gives $t_{\mathrm{mix}}(\epsilon) \leq \frac{1}{2} m^2 \lceil \lg(1/\epsilon) \rceil$, where as before $t_{\mathrm{mix}}(\epsilon)$ is the time needed to make $d_{TV}(X_t, \pi) \leq \epsilon$ (see §10.2.3).

The choice of 2 for the constant in Markov's inequality could be improved. The following lemma gives an optimized version of this argument:

Lemma 10.4.1. *Let the expected* ***coupling time****, at which two coupled processes $\{X_t\}$ and $\{Y_t\}$ starting from an arbitrary state are first equal, be T. Then $d_{TV}(X_{T_\epsilon}, Y_{T_\epsilon}) \leq \epsilon$ for $T_\epsilon \geq Te\lceil \ln(1/\epsilon) \rceil$.*

Proof. Essentially the same argument as above, but replacing 2 with a constant c to be determined. Suppose we restart the process every cT steps. Then at time t we have a total variation bounded by $c^{-\lfloor t/cT \rfloor}$. The expression $c^{-t/cT}$ is minimized by minimizing $c^{-1/c}$ or equivalently $-\ln c/c$, which occurs at $c = e$. This gives $t_{\mathrm{mix}}(\epsilon) \leq Te\lceil \ln(1/\epsilon) \rceil$. □

It's worth noting that the random walk example was very carefully rigged to make the coupling argument clean. A similar argument still works (perhaps

[9] If we know that Y_0 is uniform, then Z_0 is also uniform, and we can use this fact to get a slightly smaller bound on $\mathrm{E}[\tau]$, around $m^2/6$. But this will cause problems if we want to re-run the coupling starting from a state where X_t and Y_t have not yet converged.

with a change in the bound) for other irreducible aperiodic walks on the ring, but the details are messier.

10.4.2 Random walk on a hypercube

Start with a bit-vector of length n. At each step, choose an index uniformly at random, and set the value of the bit-vector at that index to 0 or 1 with equal probability. How long until we get a nearly-uniform distribution over all 2^n possible bit-vectors?

Here we apply the same transformation to both the X and Y vectors. It's easy to see that the two vectors will be equal once every index has been selected once. The waiting time for this to occur is just the waiting time nH_n for the coupon collector problem. We can either use this expected time directly to show that the process mixes in time $O(n \log n \log(1/\epsilon))$ as above, or apply a sharp concentration bound to the coupon collector process. It is known that (see [MU17, §5.4.1] or [MR95, §3.6.3]), $\lim_{n\to\infty} \Pr[T \geq n(\ln n + c)] = 1 - \exp(-\exp(-c)))$, so in the limit $n \ln n + n \ln \ln(1/(1-\epsilon)) = n \ln n + O(n \log(1/\epsilon))$ would seem to be enough. But this is a little tricky: we don't know from this bound alone how fast the probability converges as a function of n, so to do this right we need to look into the bound in more detail.

Fortunately, we are looking at large enough values of c that we can get a bound that is just as good using a simple argument. We have

$$\begin{aligned}\Pr[\text{there is an empty bin after } t \text{ insertions}] &\leq n\left(1 - \frac{1}{n}\right)^t \\ &\leq ne^{-t/n},\end{aligned}$$

and setting $t = n \ln n + cn$ gives a bound of e^{-c}. We can then set $c = \ln(1/\epsilon)$ to get a $\Pr[X_t \neq Y_t] \leq \epsilon$ at time $n \ln n + n \ln(1/\epsilon)$.

We can improve the bound slightly by observing that, on average, half the bits in X_0 and Y_0 are already equal; doing this right involves summing over many cases, so we won't do it.

This is an example of a Markov chain with the **rapid mixing** property: the mixing time is polylogarithmic in the number of states (2^n in this case) and $1/\epsilon$. For comparison, the random walk on the ring is not rapid mixing, because the coupling time is polynomial in $n = m$ rather than $\log n$.

10.4.3 Various shuffling algorithms

Here we have a deck of n cards, and we repeatedly apply some random transformation to the deck to converge to a stationary distribution that is uniform over all permutations of the cards (usually this is obvious by symmetry, so we won't bother proving it). Our goal is to show that the expected **coupling time** at which our deck ends up in the same permutation as an initially-stationary deck is small. Typically we do this by linking identical cards on each side of the coupling, so that each linked pair moves through the same positions with each step. When all n cards are linked, we will have achieved $X = Y$.

Lest somebody try implementing one of these shuffling algorithms, it's probably worth mentioning that they are all terrible. If you actually want to shuffle an array of values, the usual approach is to swap an element chosen uniformly at random into the first position, then shuffle the remaining $n - 1$ positions recursively. This gives a uniform shuffle in $O(n)$ steps.

10.4.3.1 Move-to-top

This is a variant of card shuffling that is interesting mostly because it gives about the easiest possible coupling argument. At each step, we choose one of the cards uniformly at random (including the top card) and move it to the top of the deck. How long until the deck is fully shuffled, i.e., until the total variation distance between the actual distribution and the stationary distribution is bounded by ϵ?

Here the trick is that when we choose a card to move to the top in the X process, we choose the same card in the Y process. It's not hard to see that this links the two cards together so that they are always in the same position in the deck in all future states. So to keep track of how well the coupling is working, we just keep track of how many cards are linked in this way, and observe that as soon as $n - 1$ are, the two decks are identical.

Note: Unlike some of the examples below, we don't consider two cards to be linked just because they are in the same position. We are only considering cards that have gone through the top position in the deck (which corresponds to some initial segment of the deck, viewed from above). The reason is that these cards never become unlinked: if we pick two cards from the initial segment, the cards above them move down together. But deeper cards that happen to match might become separated if we pull a card from one deck that is above the matched pair while its counterpart in the other deck is below the matched pair.

Given k cards linked in this way, the probability that the next step links another pair of cards is exactly $(n-k)/n$. So the expected time until we get $k+1$ cards is $n/(n-k)$, and if we sum these waiting times for $k = 0 \ldots n-1$, we get nH_n, the waiting time for the coupon collector problem. So the bound on the mixing time is the same as for the random walk on a hypercube.

10.4.3.2 Random exchange of arbitrary cards

Here we pick two cards uniformly and independently at random and swap them. (Note there is a $1/n$ chance they are the same card; if we exclude this case, the Markov chain has period 2.) To get a coupling, we reformulate this process as picking a random card and a random location, and swapping the chosen card with whatever is in the chosen location in both the X and Y processes.

First let's observe that the number of linked cards never decreases. Let x_i, y_i be the position of card i in each process, and suppose $x_i = y_i$. If neither card i nor position x_i is picked, i doesn't move, and so it stays linked. If card i is picked, then both copies are moved to the same location; it stays linked. If position x_i is picked, then it may be that i becomes unlinked; but this only happens if the card j that is picked has $x_j \neq y_j$. In this case j becomes linked, and the number of linked cards doesn't drop.

Now we need to know how likely it is that we go from k to $k+1$ linked cards. We've already seen a case where the number of linked cards increases; we pick two cards that aren't linked and a location that contains cards that aren't linked. The probability of doing this is $((n-k)/n)^2$, so our total expected waiting time is $n^2 \sum (n-k)^{-2} = n^2 \sum k^{-2} \leq n^2\pi^2/6$. The final bound is $O(n^2 \log(1/\epsilon))$.

This bound is much worse that the bound for move-to-top, which is surprising. In fact, the real bound is $O(n \log n)$ with high probability, although the proof uses very different methods (see http://www.stat.berkeley.edu/~aldous/RWG/Chap7.pdf). This shows that the coupling method doesn't always give tight bounds (or perhaps we need a better coupling?).

10.4.3.3 Random exchange of adjacent cards

Suppose now that we only swap adjacent cards. Specifically, we choose one of the n positions i in the deck uniformly at random, and then swap the cards a positions i and $i+1 \pmod n$ with probability $1/2$. (The $1/2$ is there for the usual reason of avoiding periodicity.)

So now we want a coupling between the X and Y processes where each

possible swap occurs with probability $\frac{1}{2n}$ on both sides, but somehow we correlate things so that like cards are pushed together but never pulled apart. The trick is that we will use the same position i on both sides, but be sneaky about when we swap. In particular, we will aim to arrange things so that once some card is in the same position in both decks, both copies move together, but otherwise one copy changes its position by ± 1 relative to the other with a fixed probability $\frac{1}{2n}$.

The coupled process works like this. Let D be the set of indices i where the same card appears in both decks at position i or at position $i+1$. Then we do:

1. For $i \in D$, swap $(i, i+1)$ in both decks with probability $\frac{1}{2n}$.
2. For $i \notin D$, swap $(i, i+1)$ in the X deck only with probability $\frac{1}{2n}$.
3. For $i \notin D$, swap $(i, i+1)$ in the Y deck only with probability $\frac{1}{2n}$.
4. Do nothing with probability $\frac{|D|}{2n}$.

It's worth checking that the total probability of all these events is $|D|/2n + 2(n - |D|)/2n + |D|/2n = 1$. More important is that if we consider only one of the decks, the probability of doing a swap at $(i, i+1)$ is exactly $\frac{1}{2n}$ (since we catch either case 1 or 2 for the X deck or 1 or 3 for the Y deck).

Now suppose that some card c is at position x in X and y in Y. If $x = y$, then both x and $x-1$ are in D, so the only way the card can move is if it moves in both decks: linked cards stay linked. If $x \neq y$, then c moves in deck X or deck Y, but not both. (The only way it can move in both is in case 1, where $i = x$ and $i + 1 = y$ or vice versa; but in this case i can't be in D since the copy of c at position x doesn't match whatever is in deck Y, and the copy at position y doesn't match what's in deck X.) In this case the distance $x - y$ goes up or down by 1 with equal probability $\frac{1}{2n}$. Considering $x - y \pmod n$, we have a "lazy" random walk that moves with probability $1/n$, with absorbing barriers at 0 and n. The worst-case expected time to converge is $n(n/2)^2 = n^3/4$, giving $\Pr\left[\text{time for } c \text{ to become linked} \geq \alpha n^3/2\right] \leq 2^{-\alpha}$ using the usual argument. Now apply the union bound to get $\Pr\left[\text{time for every } c \text{ to become linked} \geq \alpha n^3/2\right] \leq n2^{-\alpha}$ to get an expected coupling time of $O(n^3 \log n)$.

To simplify the argument, we assumed that the deck wraps around, so that we can swap the first and last card in addition to swapping physically adjacent cards. If we restrict to swapping i and $i+1$ for $i \in \{0, \ldots, n-2\}$, we get a slightly different process that converges in essentially the same time

$O(n^3 \log n)$. A result of David Bruce Wilson [Wil04]) shows both this upper bound holds and that the bound is optimal up to a constant factor.

10.4.3.4 Real-world shuffling

In real life, the handful of people who still use physical playing cards tend to use a **dovetail shuffle**, which is closely approximated by the reverse of a process where each card in a deck is independently assigned to a left or right pile and the left pile is place on top of the right pile. Coupling doesn't really help much here. Instead, the process can be analyzed using more sophisticated techniques due to Bayer and Diaconis [BD92]. The short version of the result is that $\Theta(\log n)$ shuffles are needed to randomize a deck of size n.

10.4.4 Path coupling

If the states of our Markov process are the vertices of a graph, we may be able to construct a coupling by considering a path between two vertices and showing how to shorten this path on average at each step. This technique is known as path coupling [BD97]. Typically, the graph we use will be the graph of possible transitions of the underlying Markov chain (possibly after making all edges undirected).

There are two ideas at work here. The first is that the expected distance $\mathrm{E}\left[d(X_t, Y_t)\right]$ between X_t and Y_t in the graph gives an upper bound on $\Pr\left[X_t \neq Y_t\right]$ (by Markov's inequality, since if $X_t \neq Y_t$ then $d(X_t, Y_t) \geq 1$). The second is that to show that $\mathrm{E}\left[d(X_{t+1}, Y_{t+1}) \mid \mathcal{F}_t\right] \leq \alpha \cdot d(X_t, Y_t)$ for some $\alpha < 1$, it is enough to show how to contract a single edge, that is, to show that $\mathrm{E}\left[d(X_{t+1}, Y_{t+1} \mid d(X_t, Y_t) = 1\right] \leq \alpha$. The reason is that if we have a coupling that contracts one edge, we can apply this inductively along each edge in the path to get a coupling between all the vertices in the path that still leaves each pair with expected distance at most α. The result for the whole path then follows from linearity of expectation.

Formally, instead of just looking at X^t and Y^t, consider a path of intermediate states $X^t = Z_0^t Z_1^t Z_2^t \ldots Z_m^t = Y^t$, where $d(Z_{i,t}, Z_{i+1,t}) = 1$ for each i (the vertices are adjacent in the graph). We now construct a coupling only for adjacent nodes that reduces their distance on average. The idea is that $d(X^t, Y^t) \leq \sum d(Z_i^t, Z_{i+1}^t)$, so if the distance between each adjacent pair shrinks on average, so does the total length of the path.

The coupling on each edge gives a joint conditional probability

$$\Pr\left[Z_i^{t+1} = z_i', Z_{i+1}^{t+1} = z_{i+1}' \;\middle|\; Z_i^t = z_i, Z_{i+1}^{t+1} = z_{i+1}\right].$$

We can extract from this a conditional distribution on Z_{i+1}^{t+1} given the other three variables:

$$\Pr\left[Z_{i+1}^{t+1} = z'_{i+1} \;\middle|\; Z_i^{t+1} = z'_i, Z_i^t = z_i, Z_{i+1}^t = z_{i+1}\right].$$

Multiplying these conditional probabilities together lets us compute a joint distribution on X^{t+1}, Y^{t+1} conditioned on X^t, Y^t, which is the ordinary coupling we really want.

It's worth noting that the path is entirely notional, and we don't actually keep it around around between steps of the coupled Markov chains. The only purpose of $Z_0, Z_1, \ldots, Z_m$ is to show that X and Y move closer together. Even though we could imagine that we are coalescing these nodes together to create a new path at each step (or throwing in a few extra nodes if some Z_i, Z_{i+1} move away from each other), we are really computing a fresh path between X and Y at each step and throwing it away as soon as it has done its job.

10.4.4.1 Random walk on a hypercube

As a warm-up, let's redo the argument about lazy random walks on a hypercube from §10.4.2 using path coupling. Each state X^t or Y^t is an n-bit vector, and with probability $1/2$ a step flips one of the bits chosen uniformly at random. The obvious metric d is Hamming distance: $d(x, y)$ is the number of indices i for which $x_i \neq y_i$.

For path coupling, we only need to push adjacent Z_i^t and Z_{i+1}^t. Adjacency means that there is exactly one index j at which these two bit-vectors differ. We apply the following coupling (which looks suspiciously like the more generic coupling in §10.4.2):

1. Pick a random index r.
2. If $r \neq j$, which occurs with probability $1 - 1/n$, flip position r in both Z_i^t and Z_{i+1}^t with probability $1/2$. Whether we flip or not, we get $d(Z_i^{t+1}, Z_{i+1}^{t+1}) = 1$.
3. If $r = j$, pick a new bit value b uniformly at random and set position j in both Z_i^{t+1} and Z_{i+1}^{t+1} to b. From either side alone, this looks like flipping bit j with probability $1/2$, exactly as in the original process. But now $d(Z_i^{t+1}, Z_{i+1}^{t+1}) = 0$.

Averaging over both cases gives $\mathrm{E}\left[d(Z_i^{t+1}, Z_{i+1}^{t+1}) \;\middle|\; \mathcal{F}_t\right] = 1 - 1/n$. It follows that $\mathrm{E}\left[d(X^t, Y^t) \mid \mathcal{F}_0\right] \leq (1 - 1/n)^t d(X_0, Y_0) \leq ne^{-n/t}$. Since $d(X^t, Y^t)$

is always at least 1 if it's not zero, this gives $\Pr\left[X^t \neq Y^t\right] \leq ne^{-n/t}$ by Markov's inequality, so $d_{TV}(X^t, Y^t) \leq \epsilon$ after $O(n \log \frac{n}{\epsilon})$ steps.

10.4.4.2 Sampling graph colorings

For this example, we'll look at sampling k-colorings of a graph with maximum degree Δ. We will assume that $k \geq 2\Delta + 1$.[10] For smaller values of k, it might still be the case that the chain converges reasonably quickly for some graphs, but our analysis will not show this.

Unlike the hypercube case, the path coupling we will construct might allow the distance between X^t and Y^t to rise with some probability. But the expected distance will always decrease, which is enough.

Consider the following chain on proper k-colorings of a graph with maximum degree Δ. At each step, we choose one of the n nodes v, compute the set S of colors not found on any of v's neighbors, and recolor v with a color chosen uniformly from S (which may be its original color).

Suppose $p_{ij} \neq 0$. Then i and j differ in at most one place v, and so the set S of permitted colors for each process—those not found on v's neighbors—are the same. This gives $p_{ij} = p_{ji} = \frac{1}{n \cdot |S|}$, and the detailed balance equations (10.3.1) hold when π_i is constant for all i. So we have a reversible Markov chain with a uniform stationary distribution. Now we will apply a path coupling to show that we converge to this stationary distribution reasonably quickly when k is large enough.

We'll think of colorings as vectors. Given two colorings x and y, let $d(x, y)$ be the Hamming distance between them, which is the number of nodes u for which $x_u \neq y_u$. To show convergence, we will construct a coupling that shows that $d(X^t, Y^t)$ converges to 0 over time starting from arbitrary initial points X^0 and Y^0.

A complication is that it's not immediately evident that the length of the shortest path from X^t to Y^t in the transition graph of our Markov chain is $d(X^t, Y^t)$. The problem is that it may not be possible to transform X^t into Y^t one node at a time without producing improper colorings. With enough colors, we can explicitly construct a short path between X^t and Y^t that uses only proper colorings; but for this particular process it is easier to simply extend the Markov chain to allow improper colorings, and show that

[10] Jerrum [Jer95] provided the random walk we'll be using, and Bubley and Dyer [BD97] used it as one of their examples. Mitzenmacher and Upfal [MU17, §12.5] give an analysis based on a standard coupling for a different version of the Markov chain that works for the same bound on k. More sophisticated results and a history of the problem can be found in [DGM02].

our coupling works anyway. This also allows us to start with an improper coloring for X^0 if we are particularly lazy. The stationary distribution is not affected, because if i is a proper coloring and j is an improper coloring that differs from i in exactly one place, we have $p_{ij} = 0$ and $p_{ji} \neq 0$, so the detailed balance equations hold with $\pi_j = 0$.

The natural coupling to consider given adjacent X^t and Y^t is to pick the same node and the same new color for both, provided we can do so. If we pick the one node v on which they differ, and choose a color that is not used by any neighbor (which will be the same for both copies of the process, since all the neighbors have the same colors), then we get $X^{t+1} = Y^{t+1}$; this event occurs with probability at least $1/n$. If we pick a node that is neither v nor adjacent to it, then the distance between X and Y doesn't change; either both get a new identical color or both don't.

Things get a little messier when we pick some node u adjacent to v, an event that occurs with probability at most Δ/n. Let c be the color of v in X^t, c' the color of v in Y^t, and T the set of colors that do not appear among the other neighbors of u. Let $\ell = |T| \geq k - (\Delta - 1)$.

Conditioning on choosing u to recolor, X^{t+1} picks a color uniformly from $T \setminus \{c\}$ and Y^{t+1} picks a color uniformly from $T \setminus \{c'\}$. We'd like these colors to be the same if possible, but these are not the same sets, and they aren't even necessarily the same size.

There are three cases:

1. Neither c nor c' are in T. Then X^{t+1} and Y^{t+1} are choosing a new color from the same set, and we can make both choose the same color: the distance between X and Y is unchanged.

2. Exactly one of c and c' is in T. Suppose that it's c. Then $|T \setminus \{c\}| = \ell - 1$ and $|T \setminus \{c'\}| = |T| = \ell$. Let X^{t+1} choose a new color c'' first. Then let $Y_u^{t+1} = c''$ with probability $\frac{\ell-1}{\ell}$ (this gives a probability of $\frac{1}{\ell}$ of picking each color in $T \setminus \{c\}$, which is what we want), and let $Y_u^{t+1} = c$ with probability $\frac{1}{\ell}$. Now the distance between X and Y increases with probability $\frac{1}{\ell}$.

3. Both c and c' are in T. For each c'' in $T \setminus \{c, c'\}$, let $X_u^{t+1} = Y_u^{t+1} = c''$ with probability $\frac{1}{\ell-1}$; since there are $\ell - 2$ such c'', this accounts for $\frac{\ell-2}{\ell-1}$ of the probability. Assign the remaining $\frac{1}{\ell-1}$ to $X_u^{t+1} = c', Y_u^{t+1} = c$. In this case the distance between X and Y increases with probability $\frac{1}{\ell-1}$, making this the worst case.

Putting everything together, we have a $1/n$ chance of picking a node that guarantees to reduce $d(X, Y)$ by 1, and at most a Δ/n chance of

picking a node that may increase $d(X, Y)$ by at most $\frac{1}{\ell-1}$ on average, where $\ell \geq k - \Delta + 1$, giving a maximum expected increase of $\frac{\Delta}{n} \cdot \frac{1}{k-\Delta}$. So

$$\begin{aligned}
\mathrm{E}\left[d(X^{t+1}, Y^{t+1}) - d(X^t, Y^t) \;\middle|\; d(X^t, Y^t) = 1\right] &\leq \frac{-1}{n} + \frac{\Delta}{n} \cdot \frac{1}{k-\Delta} \\
&= \frac{1}{n}\left(-1 + \frac{\Delta}{k-\Delta}\right) \\
&= \frac{1}{n}\left(\frac{-(k-\Delta)+\Delta}{k-\Delta}\right) \\
&= -\frac{1}{n}\left(\frac{k-2\Delta}{k-\Delta}\right).
\end{aligned}$$

So we get

$$\begin{aligned}
d_{TV}(X^t, Y^t) &\leq \Pr\left[X^t \neq Y^t\right] \\
&\leq \left(1 - \frac{1}{n} \cdot \frac{k-2\Delta}{k-\Delta}\right)^t \cdot \mathrm{E}\left[d(X^0, Y^0)\right] \\
&\leq \exp\left(-\frac{t}{n} \cdot \frac{k-2\Delta}{k-\Delta}\right) \cdot n.
\end{aligned}$$

For fixed k and Δ with $k > 2\Delta$, this is $e^{-\Theta(t/n)}n$, which will be less than ϵ for $t = \Omega(n(\log n + \log(1/\epsilon)))$.

10.4.4.3 Sampling independent sets

For a more complicated example of path coupling, let's try sampling independent sets of vertices on a graph $G = (V, E)$ with n vertices and m edges. If we can bias in favor of larger sets, we might even get a good independent set approximation! The fact that this is hard will console us when we find that it doesn't work.

A natural way to set up the random walk is to represent each potentially independent set as a bit vector, where 1 indicates membership in the set, and at each step we pick one of the n bits uniformly at random and set it to 0 or 1 with probability 1/2 each, provided that the resulting set is independent. If the resulting set is not independent, we stay put.[11]

It's easy to see that $d(x, y) = \|x - y\|_1$ is a bound on the length of the minimum number of transitions to get from x to y, since we can always remove all the extra ones from x and put back the extra ones in y while preserving

[11] In statistical physics, this process of making a local change with probability proportional to how much we like the result goes by the name of **Glauber dynamics**.

independence throughout the process. (This argument also shows that the Markov chain is irreducible.) It may be hard to find the exact minimum path length, so we'll use this distance instead for our path coupling.

We can easily show that the stationary distribution of this process is uniform. The essential idea is that if we can transform one independent set S into another S' by flipping a bit, then we can go back by flipping the bit the other ways. Since each transition happens with the same probability $1/n$, we get $\pi_S \cdot (1/n) = \pi_{S'} \cdot (1/n)$ and $\pi_S = \pi_{S'}$. Since we can apply this equation along a path between any two states, all states must have the same probability in the unique stationary distribution.

To prove convergence, it's tempting to start with the obvious coupling, even though it doesn't actually work. Pick the same position and value for both copies of the chain. If x and y are adjacent, then they coalesce with probability $1/n$ (both probability $1/2n$ transitions are feasible for both copies, since the neighboring nodes always have the same state). What is the probability that they diverge? We can only be prevented from picking a value if the value is 1 and some neighbor is 1. So the bad case is when $x_i = 1$, $y_i = 0$, and we attempt to set some neighbor of i to 1; in the worst case, this happens $\Delta/2n$ of the time, which is at least $1/n$ when $\Delta \geq 2$. No coalescence here!

This could be a sign that our random walk is no good, or it could be a sign that our coupling is no good. But we can avoid figuring out which is the case by using a sneakier random walk. We'll adopt the approach used in [MU17, §12.6].

Here the idea is that we pick a random edge uv, and then try to do one of the following operations, all with equal probability:

1. Set $u = v = 0$.
2. Set $u = 0$ and $v = 1$.
3. Set $u = 1$ and $v = 0$.

In each case, if the result would be a non-independent set, we instead do nothing.

Verifying that this has a uniform stationary distribution is mildly painful if we are not careful, since there may be several different transitions that move from some state x to the same state y. But for each transition (occurring with probability $\frac{1}{3m}$), we can see that there is a reverse transition that occurs with equal probability; so the detailed balance equations (10.3.1) hold with uniform probabilities. Note that we can argue this even though we don't

know what the actual stationary probabilities are, since we don't know how many independent sets our graph has.

So now what happens if we run two coupled copies of this process, where the copies differ on exactly one vertex i?

First, every neighbor of i is 0 in both processes. A transition that doesn't involve any neighbors of i will have the same effect on both processes. So we need to consider all choices of edges where one of the endpoints is either i or a neighbor j of i. In the case where the other endpoint isn't i, we'll call it k; there may be several such k.

If we choose ij and don't try to set j to one, we always coalesce the states. This occurs with probability $\frac{2}{3m}$. If we try to set i to zero and j to one, we may fail in both processes, because j may have a neighbor k that is already one; this will preserve the distance between the two processes. Similarly, if we try to set j to one as part of a change to some jk, we will also get a divergence between the two processes: in this case, the distance will actually increase. This can only happen if j has at most one neighbor k (other than i) that is already in the independent set; if there are two such k, then we can't set j to one no matter what the state of i is.

This argument suggests that we need to consider three cases for each j, depending on the number s of nodes $k \neq i$ that are adjacent to j and have $x_k = y_k = 1$. In each case, we assume $x_i = 0$ and $y_i = 1$, and that all other nodes have the same value in both x and y. (Note that these assumptions mean that any such k can't be adjacent to i, because we have $y_k = y_i = 1$.)

- $s = 0$. Then if we choose ij, we can always set i and j however we like, giving a net $-\frac{1}{m}$ expected change to the distance. However, this is compensated for by up to $d - 1$ attempts to set $j = 1$ and $k = 0$ for some k, all of which fail in one copy of the process but succeed in the other. Since k doesn't change, each of these failures adds only 1 to the distance, which becomes at most $\frac{d-1}{3m}$ total. So our total expected change in this case is at most $\frac{d-4}{3m}$.

- $s = 1$. Here attempts to set $i = 0$ and $j = 1$ fail in both processes, giving only a $-\frac{2}{3m}$ expected change after picking ij. Any change to jk fails only if we set $j = 1$, which we can only do in the x process and only if we also set $k = 0$ for the unique k that is currently one. This produces an increase in the distance of 2 with probability $\frac{1}{3m}$, exactly canceling out the decrease from picking ij. Total expected change is 0.

- $s = 2$. Now we can never set $j = 1$. So we drop $-\frac{2}{3m}$ from changes to ij and have no change in distance from updates to jk for any $k \neq i$.

Considering all three cases, in the worst case we have $\mathrm{E}\left[d(X_{t+1}, Y_{t+1} \mid X_t, Y_t\right] = d(X_t, Y_t) + \frac{\Delta-4}{3m}$. For $\Delta \leq 3$ (a pretty restrictive case), this is $-1/3m$, giving a decent enough expected coupling time of $O(m \log n)$.

Here, we've considered the case where all independent sets have the same probability. One can also bias the random walk in favor of larger independent sets by accepting increases with higher probability than decreases (as in Metropolis-Hastings); this samples independent sets of size s with probability proportional to λ^s. Some early examples of this approach are given in [LV97, LV99, DG00]. The question of exactly which values of λ give polynomial convergence times is still open; see [MWW07] for some more recent bounds.

10.4.4.4 Simulated annealing

Recall that the Metropolis-Hastings algorithm constructs a reversible Markov chain with a desired stationary distribution from any reversible Markov chain on the same states (see §10.3.4 for details.)

A variant, which generally involves tinkering with the chain while it's running, is the global optimization heuristic known as **simulated annealing** [KGV83]. Here we have some function g that we are trying to minimize. So we set $f(i) = \exp(-\alpha g(i))$ for some $\alpha > 0$. Running Metropolis-Hastings gives a stationary distribution that is exponentially weighted to small values of g; if i is the global minimum and j is some state with high $g(j)$, then $\pi(i) = \pi(j) \exp(\alpha(g(j) - g(i))$, which for large enough α goes a long way towards compensating for the fact that in most problems there are likely to be exponentially more bad j's than good i's. The problem is that the same analysis applies if i is a local minimum and j is on the boundary of some depression around i; large α means that it is exponentially unlikely that we escape this depression and find the global minimum.

The simulated annealing hack is to vary α over time; initially, we set α small, so that we get mixing time close to that of the original Markov chain. This gives us a sample that is roughly uniform, with a small bias towards states with smaller $g(i)$. After some time we increase α to force the process into better states. The hope is that by increasing α slowly, by the time we are stuck in some depression, it's a deep one—optimal or close to it. If it doesn't work, we can randomly restart and/or decrease α repeatedly to jog the chain out of whatever depression it is stuck in. How to do this effectively is deep voodoo that depends on the structure of the underlying chain and the shape of $g(i)$, so most of the time people who use simulated annealing just try it out with some generic **annealing schedule** and hope it gives some

useful result. (This is what makes it a heuristic rather than an algorithm. Its continued survival is a sign that it does work at least sometimes.)

Here are toy examples of simulated annealing with provable convergence times.

Single peak Let's suppose x is a random walk on an n-dimensional hypercube (i.e., n-bit vectors where we set 1 bit at a time), $g(x) = |x|$, and we want to maximize g. Now a transition that increase $|x|$ is accepted always and a transition that decreases $|x|$ is accepted only with probability $e^{-\alpha}$. For large enough α, this puts a constant fraction of π on the single peak at $x = \mathbf{1}$; the observation is that that there are only $\binom{n}{k} \leq n^k$ points with k zeros, so the total weight of all points is at most $\pi(\mathbf{1}) \sum_{k\geq 0} n^k \exp(-\alpha k) = \pi(\mathbf{1}) \sum \exp(\ln n - \alpha)^k = \pi(\mathbf{1})/(1 - \exp(\ln n - \alpha)) = \pi(\mathbf{1}) \cdot O(1)$ when $\alpha > \ln n$, giving $\pi(\mathbf{1}) = \Omega(1)$ in this case.

So what happens with convergence? Let $p = \exp(-\alpha)$. Let's try doing a path coupling between two adjacent copies x and y of the Metropolis-Hastings process, where we first pick a bit to change, then pick a value to assign to it, accepting the change in both processes if we can. The expected change in $|x - y|$ is then $(1/2n)(-1 - p)$, since if we pick the bit where x and y differ, we have probability $1/2n$ of setting both to 1 and probability $p/2n$ of setting both to 0, and if we pick any other bit, we get the same distribution of outcomes in both processes. This gives a general bound of $\mathrm{E}\left[|X_{t+1} - Y_{t+1}| \mid |X_t - Y_t|\right] \leq (1 - (1+p)/2n)|X_t - Y_t|$, from which we have $\mathrm{E}\left[|X_t - Y_t|\right] \leq \exp(-t(1+p)/2n)\,\mathrm{E}\left[|X_0 - Y_0|\right] \leq n \exp(-t(1+p)/2n)$. So after $t = 2n/(1+p)\ln(n/\epsilon)$ steps, we expect to converge to within ϵ of the stationary distribution in total variation distance. This gives an $O(n \log n)$ algorithm for finding the peak.

This is kind of a silly example, but if we suppose that g is better disguised (for example, $g(x)$ could be $|x \oplus r|$ where r is a random bit vector), then we wouldn't really expect to do much better than $O(n)$. So $O(n \log n)$ is not bad for an algorithm with no brains at all.

Somewhat smooth functions Now we'll let $g : 2^n \to \mathbb{N}$ be some arbitrary Lipschitz function, that is, a function with the property that $|g(x) - g(y)| \leq |x - y|$, and ask for what values of $p = e^{-\alpha}$ the Metropolis-Hastings walk with $f(i) = e^{-\alpha g(i)}$ can be shown to converge quickly. Given adjacent states x and y, with $x_i \neq y_i$ but $x_j = y_j$ for all $j \neq i$, we still have a probability of at least $(1+p)/2n$ of coalescing the states by setting $x_i = y_i$. But now there is a possibility that if we try to move to $(x[j/b], y[j/b])$ for some j and

b, that x rejects while y does not or vice versa (note if $x_j = y_j = b$, we don't move in either copy of the process). Conditioning on j and b, this occurs with probability $1-p$ precisely when $x[j/b] < x$ and $y[j/b] \geq y$ or vice versa, giving an expected increase in $|x-y|$ of $(1-p)/2n$. We still get an expected net change of $-2p/2n = -p/n$ provided there is only one choice of j and b for which this occurs. So we converge in time $\tau(\epsilon) \leq (n/p)\log(n/\epsilon)$ in this case.[12]

One way to think of this is that the shape of the neighborhoods of nearby points is similar. If I go up in a particular direction from point x, it's very likely that I go up in the same direction from some neighbor y of x.

If there are more bad choices for j and b, then we need a much larger value of p: the expected net change is now $(k(1-p)-(1+p))/2n = (k-1-(k+1)p)/2n$, which is only negative if $p > (k-1)/(k+1)$. This gives much weaker pressure towards large values of g, which still tends to put us in high neighborhoods but creates the temptation to fiddle with α to try to push us even higher once we think we are close to a peak.

10.5 Spectral methods for reversible chains

(See also http://www.stat.berkeley.edu/~aldous/RWG/Chap3.pdf, from which many of the details in the notes below are taken.)

The problem with coupling is that (a) it requires cleverness to come up with a good coupling; and (b) in many cases, even that doesn't work—there are problems for which no coupling that only depends on current and past transitions coalesces in a reasonable amount of time.[13] When we run into these problems, we may be able to show convergence instead using a linear-algebraic approach, where we look at the eigenvalues of the transition matrix of our Markov chain. This approach works best for reversible Markov chains, where $\pi_i p_{ij} = \pi_j p_{ij}$ for all states i and j and some distribution π.

10.5.1 Spectral properties of a reversible chain

Suppose that P is the transition matrix of an irreducible, reversible Markov chain. Then it has a unique stationary distribution π that is a left **eigenvector** corresponding to the **eigenvalue** 1, which just means that $\pi P = 1\pi$.

[12] You might reasonably ask if such functions g exist. One example is $g(x) = (x_1 \oplus x_2) + \sum_{i>2} x_i$.

[13] A coupling that doesn't require looking into the future is called a **causal coupling**. An example of a Markov chain for which causal couplings are known not to work is the one used for sampling perfect matchings in bipartite graphs as described in §10.6.4.4 [KR99].

For irreducible aperiodic chains, P will have a total of n real eigenvalues $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n$, where $\lambda_1 = 1$ and $\lambda_n > -1$. (This follows for symmetric chains from the **Perron-Frobenius theorem**, which we will not attempt to prove here; for general irreducible aperiodic reversible chains, we will show below that we can reduce to the symmetric case.) Each eigenvalue λ_i has a corresponding eigenvector u^i, a nonzero vector that satisfies $u^i P = \lambda_i u^i$. In Markov chain terms, these eigenvectors correspond to deviations from the stationary distribution that shrink over time.

For example, the transition matrix

$$S = \begin{bmatrix} p & q \\ q & p \end{bmatrix}$$

corresponding to a Markov chain on two states that stays in the same state with probability p and switches to the other state with probability $q = 1 - p$ has eigenvectors $u_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $u_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}$ with corresponding eigenvalues $\lambda_1 = 1$ and $\lambda_2 = p - q$, as shown by computing

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} p & q \\ q & p \end{bmatrix} = \begin{bmatrix} p+q & q+p \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} p & q \\ q & p \end{bmatrix} = \begin{bmatrix} p-q & q-p \end{bmatrix} = (p-q) \cdot \begin{bmatrix} 1 & -1 \end{bmatrix}.$$

10.5.2 Analysis of symmetric chains

To make our life easier, we will assume that in addition to being reversible, our Markov chain has a uniform stationary distribution. Then $\pi_i = \pi_j$ for all i and j, and so reversibility implies $p_{ij} = p_{ji}$ for all i and j as well, meaning that the transition matrix P is symmetric. Symmetric matrices have the very nice property that their eigenvectors are all orthogonal (this is the **spectral theorem**), which allows for a very straightforward decomposition of the distribution on the initial state (represented as a vector x_0) as a linear combination of the eigenvectors. For example, if initially we put all our weight on state 1, we get $x_0 = \begin{bmatrix} 1 & 0 \end{bmatrix} = \frac{1}{2}u^1 + \frac{1}{2}u^2$.

If we now take a step in the chain, we multiply x by P. We can express

this as

$$\begin{aligned} xP &= \left(\frac{1}{2}u^1 + \frac{1}{2}u^2\right)P \\ &= \frac{1}{2}u^1P + \frac{1}{2}u^2P \\ &= \frac{1}{2}\lambda_1 u^1 + \frac{1}{2}\lambda_2 u^2. \end{aligned}$$

This uses the defining property of eigenvectors, that $u^iP = \lambda_i u^i$.

In general, if $x = \sum_i a_i u^i$, then $xP = \sum_i a_i\lambda_i u^i$ and $xP^t = \sum a_i\lambda_i^t u^i$. For any eigenvalue λ_i with $|\lambda_i| < 1$, λ_i^t goes to zero in the limit. So only those eigenvectors with $\lambda_i = 1$ survive. For irreducible aperiodic chains, these consist only of the stationary distribution $\pi = u^i/\|u^i\|_1$. For reducible or periodic chains, there may be some additional eigenvalues with $|\lambda_i| = 1$, but the only possibility that arises for an irreducible reversible chain is $\lambda_n = -1$, corresponding to a chain with period 2.[14]

Assuming that $|\lambda_2| \geq |\lambda_n|$, as t grows large λ_2^t will dominate the other smaller eigenvalues, and so the size of λ_2 will control the rate of convergence of the underlying Markov process.

This assumption is always true for lazy walks that stay put with probability 1/2, because all eigenvalues of a lazy walk are non-negative. The reason is that any such walk has transition matrix $\frac{1}{2}(P + I)$, where I is the identity matrix and P is the transition matrix of the unlazy version of the walk. If $xP = \lambda x$ for some x and λ, then $x\Big(\frac{1}{2}(P + I)\Big) = \Big(\frac{1}{2}(\lambda + 1)\Big)x$. This means that x is still an eigenvector of the lazy walk, and its corresponding eigenvalue is $\frac{1}{2}(\lambda + 1) \geq \frac{1}{2}((-1) + 1) \geq 0$.

But what does having small λ_2 mean for total variation distance? If x^t is a vector representing the distribution of our position at time t, then $d_{TV} = \frac{1}{2}\sum_{i=1}^n |x_i^t - \pi_i| = \frac{1}{2}\|x^t - \pi\|_1$. But we know that $x^0 = \pi + \sum_{i=2}^n c_i u^i$ for some coefficients c_i, and $x^t = \pi + \sum_{i=2}^n \lambda_i^t c_i u^i$. So we are looking for a bound on $\|\sum_{i=2}^n \lambda_i^t c_i u^i\|_1$.

It turns out that it is easier to get a bound on the ℓ_2 norm $\|\sum_{i=2}^n \lambda_i^t c_i u^i\|_2$. Here we use the fact that the eigenvectors are orthogonal. This means that the Pythagorean theorem holds and $\|\sum_{i=2}^n \lambda_i^t c_i u^i\|_2^2 = \sum_{i=2}^n \lambda_i^{2t} c_i^2 \|u^i\|_2^2 =$

[14] Chains with periods greater than 2 (which are never reversible) have pairs of complex-valued eigenvalues that are roots of unity, which happen to cancel out to only produce real probabilities in vP^t. Chains that aren't irreducible will have one eigenvector with eigenvalue 1 for each final component; the stationary distributions of these chains are linear combinations of these eigenvectors (which are just the stationary distributions on each component).

$\sum_{i=2}^n \lambda_i^{2t} c_i^2$ if we normalize each u^i so that $\|u^i\|_2^2 = 1$. But then

$$\begin{aligned}\left\|x^t - \pi\right\|_2^2 &= \sum_{i=2}^n \lambda_i^{2t} c_i^2 \\ &\le \sum_{i=2}^n \lambda_2^{2t} c_i^2 \\ &= \lambda_2^{2t} \sum_{i=2}^n c_i^2 \\ &= \lambda_2^{2t} \|x^0 - \pi\|_2^2.\end{aligned}$$

Now take the square root of both sides to get

$$\left\|x^t - \pi\right\|_2 =< \lambda_2^t \left\|x^0 - \pi\right\|.$$

To translate this back into d_{TV}, we use the inequality

$$\|x\|_2 \le \|x\|_1 \le \sqrt{n} \cdot \|x\|_2,$$

which holds for any n-dimensional vector x. Because $\|x^0\|_1 = \|\pi\|_1 = 1$, $\|x^0 - \pi\|_1 \le 2$ by the triangle inequality, which also gives $\|x^0 - \pi\|_2 \le 2$. So

$$\begin{aligned}d_{TV}(x^t, \pi) &= \frac{1}{2}\|x^t - \pi\|_1 \\ &\le \frac{\sqrt{n}}{2}\lambda_2^t \|x^0 - \pi\|_2 \\ &\le \lambda_2^t \sqrt{n}.\end{aligned}$$

to get a bound on $\|x - \pi\|_1$.

If we want to get $d_T V(x^t, n) \le \epsilon$, we will need $t \ln(\lambda_2) + \ln\sqrt{n} \le \ln \epsilon$ or $t \ge \frac{1}{\ln(1/\lambda_2)}\left(\frac{1}{2}\ln n + \ln(1/\epsilon)\right)$.

The factor $\frac{1}{\ln(1/\lambda_2)} = \frac{1}{-\ln \lambda_2}$ can be approximated by $\tau_2 = \frac{1}{1-\lambda_2}$, which is called the **mixing rate** or **relaxation time** of the Markov chain. Indeed, our old friend $1 + x \le e^x$ implies $\ln(1 + x) \le x$, which gives $\ln \lambda_2 \le \lambda_2 - 1$ and thus $\frac{1}{-\ln \lambda_2} \le \frac{1}{1-\lambda_2} = \tau_2$. So $\tau_2\left(\frac{1}{2}\ln n + \ln(1/\epsilon)\right)$ gives a conservative estimate on the time needed to achieve $d_{TV}(x^t, \pi) \le \epsilon$ starting from an arbitrary distribution.

It's worth comparing this to the mixing time $t_{\text{mix}} = \arg\min_t d_{TV}(x^t, \pi) \le 1/4$ that we used with coupling. With τ_2, we have to throw in an extra factor of $\frac{1}{2}\ln n$ to get the bound down to 1, but after that the bound continues to

drop by a factor of e every τ_2 steps. With t_{mix}, we have to go through another t_{mix} steps for each factor of 2 improvement in total variation distance, even if the initial drop avoids the log factor. Since we can't always tell whether coupling arguments or spectral arguments will do better for a particular chain, and since convergence bounds are often hard to obtain no matter how many techniques we throw at them, we will generally be happy with any reasonable bound on either τ_2 or t_{mix}.

10.5.3 Analysis of asymmetric chains

If the stationary distribution is not uniform, then in general the transition matrix will not be symmetric. We can make it symmetric by scaling the probability of being in the i-th state by $\pi_i^{-1/2}$. The idea is to decompose our transition matrix P as $\Pi A \Pi^{-1}$, where Π is a diagonal matrix with $\Pi_{ii} = \pi_i^{-1/2}$, and $A_{ij} = \sqrt{\frac{\pi_i}{\pi_j}} P_{ij}$. Then A is symmetric, because

$$\begin{aligned} A_{ij} &= \sqrt{\frac{\pi_i}{\pi_j}} P_{ij} \\ &= \sqrt{\frac{\pi_i}{\pi_j}} \frac{\pi_j}{\pi_i} P_{ji} \\ &= \sqrt{\frac{\pi_j}{\pi_i}} P_{ji} \\ &= A_{ji}. \end{aligned}$$

This means in particular that A has orthogonal eigenvectors $u^1, u^2, \ldots, u^n$ with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. These eigenvalues carry over to $P = \Pi A \Pi^{-1}$, since $(u^i \Pi^{-1})P = u^i \Pi^{-1} \Pi A \Pi^{-1} = u^i A \Pi^{-1} = \lambda_i u^i \Pi^{-1}$.

So now given an initial distribution x_0, we can first pass it through Π and then apply the same reasoning as before to show that $\|\pi\Pi - x\Pi\|_2$ shrinks by λ_2 with every step of the Markov chain. The difference now is that the initial distance is affected by the scaling we did in Π^{-1}; so instead of getting $d_{TV}(x^t, \pi) \leq \lambda_2^t \sqrt{n}$, we get $d_{TV}(x^t, \pi) \leq \lambda_2^t (\pi_{\min})^{-1/2}$, where $\pi_{\min}$ is the smallest probability of any single node in the stationary distribution π. The bad initial x_0 in this case is the one that puts all its weight on this node, since this maximizes its distance from π after scaling.

(A uniform π is a special case of this, since when $\pi_i = 1/n$ for all i, $\pi_{\min}^{-1/2} = (1/n)^{-1/2} = \sqrt{n}$.)

For more details on this, see [AF01, §3.4].

So now we just need a tool for bounding λ_2. For a small chain with a known transition matrix, we can just feed it to our favorite linear algebra library, but most of the time we will not be able to construct the matrix explicitly. So we need a way to bound λ_2 indirectly, in terms of other structural properties of our Markov chain.

10.6 Conductance

The **conductance** or **Cheeger constant** $\Phi(S)$ of a set S of states in a Markov chain is

$$\Phi(S) = \frac{\sum_{i \in S, j \notin S} \pi_i p_{ij}}{\pi(S)}. \tag{10.6.1}$$

This is the probability of leaving S on the next step starting from the stationary distribution conditioned on being in S. The conductance is a measure of how easy it is to escape from a set. It can also be thought of as a weighted version of edge expansion.

The conductance of a Markov chain as a whole is obtained by taking the minimum of $\Phi(S)$ over all S that occur with probability at most $1/2$:

$$\Phi = \min_{0 < \pi(S) \leq 1/2} \Phi(S). \tag{10.6.2}$$

The usefulness of conductance is that it bounds λ_2:

Theorem 10.6.1. *In a reversible Markov chain,*

$$1 - 2\Phi \leq \lambda_2 \leq 1 - \Phi^2/2. \tag{10.6.3}$$

The bound (10.6.3) is known as the **Cheeger inequality**. We won't attempt to prove it here, but the intuition is that in order for a reversible chain not to mix, it has to get stuck in some subset of the states. Having high conductance prevents this disaster, while having low conductance causes it.

For lazy walks we always have $\lambda_2 = \lambda_{\max}$, and so we can convert (10.6.3) to a bound on the relaxation time:

Corollary 10.6.2.

$$\frac{1}{2\Phi} \leq \tau_2 \leq \frac{2}{\Phi^2}. \tag{10.6.4}$$

In other words, high conductance implies low relaxation time and vice versa, up to squaring.

10.6.1 Easy cases for conductance

For very simple Markov chains we can compute the conductance directly. Consider a lazy random walk on a cycle. Any proper subset S has at least two outgoing edges, each of which carries a flow of $1/4n$, giving $\Phi_S \geq (1/2n)/\pi(S)$. If we now take the minimum of Φ_S over all S with $\pi(S) \leq 1/2$, we get $\phi \geq 1/n$, which gives $\tau_2 \leq 2n^2$. This is essentially the same bound as we got from coupling.

Here's a slightly more complicated chain. Take two copies of K_n, where n is odd, and join them by a path with n edges. Now consider Φ_S for a lazy random walk on this graph where S consists of half the graph, split at the edge in the middle of the path. There is a single outgoing edge uv, with $\pi(u) = d(u)/2|E| = 2/(2n(n-1)n/2 + n) = 2n^{-2}$ and $p_{uv} = 1/4$, for $\pi(u)p_{uv} = n^{-2}/2$. By symmetry, we have $\pi(S) \to 1/2$ as $n \to \infty$, giving $\Phi_S \to n^{-2}(1/2)/(1/2) = n^{-2}$. So we have $n^2/2 \leq \tau_2 \leq 2n^4$.

How does this compare to the actual mixing time? In the stationary distribution, we have a constant probability of being in each of the copies of K_n. Suppose we start in the left copy. At each step there is a $1/n$ chance that we are sitting on the path endpoint. We then step onto the path with probability $1/n$, and reach the other end before coming back with probability $1/n$. So (assuming we can make this extremely sloppy handwaving argument rigorous) it takes at least n^3 steps on average before we reach the other copy of K_n, which gives us a rough estimate of the mixing time of $\Theta(n^3)$. In this case the exponent is exactly in the middle of the bounds derived from conductance.

Curiously, it's not hard to argue that this is in fact the worst possible conductance we can get from a lazy random walk. Consider an arbitrary connected graph G and any subset S of $V(G)$. Then S has at least one outgoing edge uv, and we take it with probability $\pi_u p_{uv} = \frac{d(u)}{\sum_w d(w)} \cdot \frac{1}{2d(u)} = \frac{1}{\sum_w d(w)} = \frac{1}{2|E|} \geq \frac{1}{n(n-1)}$. Assuming $\pi(S) \leq 1/2$, this gives $\Phi(S) \geq \frac{2}{n(n-1)} = \Theta(1/n^2)$. This immediately gives that a lazy random walk on *any* graph converges in time $\widetilde{O}(n^4)$.

10.6.2 Edge expansion using canonical paths

(Here and below we are mostly following the presentation of Guruswami [Gur00], but with slightly different examples.)

For more complicated Markov chains, it is helpful to have a tool for bounding conductance that doesn't depend on intuiting what sets have the smallest boundary. The **canonical paths** [JS89] technique does this by

assigning a unique path γ_{xy} from each state x to each state y in a way that doesn't send too many paths across any one edge. So if we have a partition of the state space into sets S and T, then there are $|S| \cdot |T|$ paths from states in S to states in T, and since (a) every one of these paths crosses an S–T edge, and (b) each S–T edge carries at most ρ paths, there must be at least $|S| \cdot |T|/\rho$ edges from S to T. Note that because there is no guarantee we chose good canonical paths, this is only useful for getting lower bounds on conductance—and thus upper bounds on mixing time—but this is usually what we want.

Let's start with a small example. Let $G = C_n \square C_m$, the $n \times m$ torus. A lazy random walk on this graph moves north, east, south, or west with probability 1/8 each, wrapping around when the coordinates reach n or m. Since this is a random walk on a regular graph, the stationary distribution is uniform. What is the relaxation time?

Intuitively, we expect it to be $O(\max(n, m)^2)$, because we can think of this two-dimensional random walk as consisting of two one-dimensional random walks, one in the horizontal direction and one in the vertical direction, and we know that a random walk on a cycle mixes in $O(n^2)$ time. Unfortunately, the two random walks are not independent: every time I take a horizontal step is a time I am not taking a vertical step. We *can* show that the expected coupling time is $O(n^2 + m^2)$ by running two sequential instances of the coupling argument for the cycle, where we first link the two copies in the horizontal direction and then in the vertical direction. So this gives us one bound on the mixing time. But what happens if we try to use conductance?

Here it is probably easiest to start with just a cycle. Given points x and y on C_n, let the canonical path γ_{xy} be a shortest path between them, breaking ties so that half the paths go one way and half the other. Then each each is crossed by exactly k paths of length k for each $k = 1 \dots (n/2 - 1)$, and at most $n/4$ paths of length $n/2$ (0 if n is odd), giving a total of $\rho \leq \binom{n/2}{2} + \frac{n}{4} = \frac{(n/2-1)(n/2)}{2} + \frac{n}{4} = n^2/8$ paths across the edge.

If we now take an S–T partition where $|S| = m$, we get at least $m(n - m)/\rho = 8m(n - m)/n^2$ S–T edges. This peaks at $m = n/2$, where we get 2 edges—exactly the right number—and in general when $m \leq n/2$ we get at least $8m(n/2)/n^2 = 4m/n$ outgoing edges, giving a conductance $\Phi_S \geq (1/4n)(4m/n)/(m/n) = 1/n$.

This is essentially what we got before, except we have to divide by 2 because we are doing a lazy walk. Note that for small m, the bound is a gross underestimate, since we know that every nonempty proper subset has at least 2 outgoing edges.

Now let's go back to the torus $C_n \square C_m$. Given x and y, define γ_{xy} to be the L-shaped path that first changes x_1 to y_1 by moving the shortest distance vertically, then changes x_2 to y_2 by moving the shortest distance horizontally. For a vertical edge, the number of such paths that cross it is bounded by $n^2m/8$, since we get at most $n^2/8$ possible vertical path segments and for each such vertical path segment there are m possible horizontal destinations to attach to it. For a horizontal edge, n and m are reversed, giving $m^2n/8$ paths. To make our life easier, let's assume $n \geq m$, giving a maximum of $\rho = n^2m/8$ paths over any edge.

For $|S| \leq nm/2$, this gives at least

$$\frac{|S| \cdot |G \setminus S|}{\rho} \geq \frac{|S|(nm/2)}{n^2m/8} = \frac{4|S|}{n}$$

outgoing edges. We thus have

$$\Phi(S) \geq \frac{1}{8nm} \cdot \frac{4|S|/n}{|S|/nm} = \frac{1}{2n}.$$

This gives $\tau_2 \leq 2/\Phi^2 \leq 8n^2$. Given that we assumed $n \geq m$, this is essentially the same $O(n^2 + m^2)$ bound that we would get from coupling.

10.6.3 Congestion

For less symmetric chains, we weight paths by the probabilities of their endpoints when counting how many cross each edge, and treat the flow across the edge as a capacity. This gives the **congestion** of a collection of canonical paths $\Gamma = \{\gamma_{xy}\}$, which is computed as

$$\rho(\Gamma) = \max_{uv \in E} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y,$$

and we define $\rho = \min_\Gamma \rho(\Gamma)$.

The intuition here is that the congestion bounds the ratio between the canonical path flow across an edge and the Markov chain flow across the edge. If the congestion is not too big, this means that any cut that has a lot of canonical path flow across it also has a lot of Markov chain flow. When $\pi(T) \geq \frac{1}{2}$, the total canonical path flow $\pi(S)\pi(T)$ is at least $\frac{1}{2}\pi(S)$. This means that when $\pi(S)$ is large but less than $\frac{1}{2}$, the Markov chain flow leaving S is also large. This gives a lower bound on conductance.

Formally, we have the following lemma:

Lemma 10.6.3. *For any reversible aperiodic irreducible Markov chain,*

$$\Phi \geq \frac{1}{2\rho} \tag{10.6.5}$$

from which it follows that

$$\tau_2 \leq 8\rho^2. \tag{10.6.6}$$

Proof. Pick some set of canonical paths Γ with $\rho(\Gamma) = \rho$. Now pick some S–T partition with $\pi(S) \leq 1/2$ and $\phi(S) = \phi$. Consider a flow where we route $\pi(x)\pi(y)$ units of flow along each path γ_{xy} with $x \in S$ and $y \in T$. This gives a total flow of $\pi(S)\pi(T) \geq \pi(S)/2$. We are going to show that we need a lot of capacity across the S–T cut to carry this flow, which will give the lower bound on conductance.

For any S–T edge uv, we have

$$\frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \leq \rho$$

or

$$\sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \leq \rho \cdot \pi_u p_{uv}.$$

Since each S–T path crosses at least one S–T edge, we have

$$\begin{aligned}
\pi(S)\pi(T) &= \sum_{x \in S, y \in T} \pi_x \pi_y \\
&\leq \sum_{u \in S, v \in T, uv \in E} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \\
&\leq \sum_{u \in S, v \in T, uv \in E} \rho \pi_u p_{uv}. \\
&= \rho \sum_{u \in S, v \in T, uv \in E} \pi_u p_{uv}.
\end{aligned}$$

But then

$$\begin{aligned}
\Phi(S) &= \frac{\sum_{u \in S, t \in T, uv \in E} \pi_u p_{uv}}{\pi_S} \\
&\geq \frac{\pi(S)\pi(T)/\rho}{\pi(S)} \\
&= \frac{\pi(T)}{\rho} \\
&\geq \frac{1}{2\rho}.
\end{aligned}$$

To get the bound on τ_2, use (10.6.4) to compute $\tau_2 \leq 2/\Phi^2 \leq 8\rho^2$. □

10.6.4 Examples

Here are some more examples of applying canonical paths.

10.6.4.1 Lazy random walk on a line

Consider a lazy random walk on a line with reflecting barriers at 0 and $n-1$. Here $\pi_x = \frac{1}{n}$ for all x. There aren't a whole lot of choices for canonical paths; the obvious choice for γ_{xy} with $x < y$ is $x, x+1, x+2, \ldots, y$. This puts $(n/2)^2$ paths across the middle edge (which is the most heavily loaded), each of which has weight $\pi_x \pi_y = n^{-2}$. So the congestion is $\frac{1}{(1/n)(1/4)}(n/2)^2 n^{-2} = n$, giving a mixing time of at most $8n^2$. This is a pretty good estimate.

10.6.4.2 Random walk on a hypercube

Let's try a more complicated example: the random walk on a hypercube from §10.4.2. Here at each step we pick some coordinate uniformly at random and set it to 0 or 1 with equal probability; this gives a transition probability $p_{uv} = \frac{1}{2n}$ whenever i and j differ by exactly one bit. A plausible choice for canonical paths is to let γ_{xy} use **bit-fixing routing**, where we change bits in x to the corresponding bits in y from left to right. To compute congestion, pick some edge uv, and let k be the bit position in which u and v differ. A path γ_{xy} will cross uv if $u_k \ldots u_n = x_k \ldots x_n$ (because when we are at u we haven't fixed those bits yet) and $v_1 \ldots v_k = y_1 \ldots y_k$ (because at v we have fixed all of those bits). There are 2^{k-1} choices of $x_1 \ldots x_{k-1}$ consistent with the first condition and 2^{n-k} choices of $y_{k+1} \ldots y_n$ consistent with the second, giving exactly 2^{n-1} total paths γ_{xy} crossing uv. Since each path occurs with weight $\pi_x \pi_y = 2^{-2n}$, and the flow across uv is $\pi_u p_{uv} = 2^{-n}\frac{1}{2n}$, we can calculate the congestion

$$\begin{aligned}
\rho(\Gamma) &= \max_{uv \in E} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y \\
&= \frac{1}{2^{-n}/2n} \cdot 2^{n-1} \cdot 2^{-2n} \\
&= n.
\end{aligned}$$

This gives a relaxation time $\tau_2 \le 8\rho^2 = 8n^2$, which when we account for the large state space gives $t_{\text{mix}}(\epsilon) \le 8n^2\left(\frac{1}{2}\ln 2n^n + \ln(1/\epsilon)\right) = O(n^3)$. In this case the bound is substantially worse than what we previously proved using coupling.

The fact that the number of canonical paths that cross a particular edge is exactly one half the number of nodes in the hypercube is not an accident: if we look at what information we need to fill in to compute x and y from u and v, we need (a) the part of x we've already gotten rid of, plus (b) the part of y we haven't filled in yet. If we stitch these two pieces together, we get all but one of the n bits we need to specify a node in the hypercube, the missing bit being the bit we flip to get from u to v. This sort of thing shows up often in conductance arguments where we build our canonical paths by fixing a structure one piece at a time.

10.6.4.3 Matchings in a graph

A **matching** in a graph $G = (V, E)$ is a subset of the edges with no two elements adjacent to each other; equivalently, it's a subgraph that includes all the vertices in which each vertex has degree at most 1. We can use a random walk to sample matchings from an arbitrary graph uniformly.

Here is the random walk. Let S_t be the matching at time t. At each step, we choose an edge $e \in E$ uniformly at random, and flip a coin to decide whether to include it or not. If the coin comes up tails, set $S_{t+1} = S_t \backslash \{e\}$ (this may leave $S_{t+1} = S_t$ if S_t already omitted e); otherwise, set $S_{t+1} = S_t \cup \{e\}$ unless one of the endpoints of e is already incident to an edge in S_t, in which case set $S_{t+1} = S_t$.

Because this chain contains many self-loops, it's aperiodic. It's also straightforward to show that any transition between two adjacent matchings occurs with probability exactly $\frac{1}{2m}$, where $m = |E|$, and thus that the chain is reversible with a uniform stationary distribution. We'd like to bound the congestion of the chain to show that it converges in a reasonable amount of time.

Let N be the number of matchings in G, and let $n = |V|$ and $m = |E|$ as usual. Then $\pi_S = 1/N$ for all S and $\pi_S p_{ST} = \frac{1}{2Nm}$. Our congestion for any transition ST will then be $2NmN^{-2} = 2m/N$ times the number of paths that cross ST; ideally this number of paths will be at most N times some small polynomial in n and/or m.

Suppose we are trying to get from some matching X to another matching Y. The graph $X \cup Y$ has maximum degree 2, so each of its connected components is either a path or a cycle; in addition, we know that the edges in each of these paths or cycles alternate whether they come from X or Y, which among other things implies that the cycles are all even cycles.

We can transform X to Y by processing these components in a methodical way: first order the components (say, by the increasing identity of the smallest

vertex in each), then for each component replace the X edges with Y edges. If we do this cleverly enough, we can guarantee that for any transition ST, the set of edges $(X \cup Y) \setminus (S \cup T)$ always consists of a matching plus at most two extra edges, and that S, T, $(X \cup Y) \setminus (S \cup T)$ are enough to reconstruct X and Y. Since there are at most Nm^2 choices of $(X \cup Y) \setminus (S \cup T)$, this will give at most Nm^2 canonical paths across each transition.[15]

Here is how we do the replacement within a component. If C is a cycle with k vertices, order the vertices $v_0, \ldots, v_{k-1}$ such that v_0 has the smallest index in C and $v_0v_1 \in X$. Then $X \cap C$ consists of the even-numbered edges $e_{2i} = v_{2i}v_{2+1}$ and $Y \cap C$ consists of the odd-numbered edges $e_{2i+1} = v_{2i+1}v_{2i+2}$, where we take $v_k = v_0$. We now delete e_0, then alternate between deleting e_{2i} and adding e_{2i-1} for $i = 1 \ldots k/2 - 1$. Finally we add e_{k-1}.

The number of edges in C at each step in this process is always one of $k/2$, $k/2 - 1$, or $k/2 - 2$, and since we always add or delete an edge, the number of edges in $C \cap (S \cup T)$ for any transition will be at least $k/2 - 1$, which means that the total degree of the vertices in $C \cap (S \cup T)$ is at least $k - 2$. We also know that SUT is a matching, since it's either equal to S or equal to T. So there are at least $k - 2$ vertices in $C \cap (S \cup T)$ with degree 1, and none with degree 2, leaving at most 2 vertices with degree 0. In the complement $C \setminus (S \cup T)$, this becomes at most two vertices with degree 2, with the rest having degree 1. If these vertices are adjacent, removing the edge between them leaves a matching; if not, removing one edge adjacent to each does so. So two extra edges are enough.

A similar argument works for paths, which we will leave as an exercise for the reader.

Now suppose we know S, T, and $(XUY) \setminus (S \cup T)$. We can compute $X \cup Y = ((X \cup Y) \setminus (S \cup T)) \cup (S \cup T)$; this means we can reconstruct the graph $X \cup Y$, identify its components, and so on. We know which component C we are working on because we know which edge changes between S and T. For any earlier component C', we have $C' \cap S = C' \cap Y$ (since we finished already), and similarly $C' \setminus S = C' \cap X$. For components C' we haven't reached yet, the reverse holds, $C' \cap S = C \cap X$ and $C' \setminus S = C \cap Y$. This determines the edges in both X and Y for all edges not in C.

For C, we know from $C \cap (X \cup Y)$ which vertex is v_0. In a cycle, whenever we remove an edge, its lower-numbered endpoint is always an even distance from v_0, and similarly when we add an edge, its lower-numbered endpoint is always an odd distance from v_0. So we can orient the cycle and tell

[15] We could improve the constant a bit by using $N\binom{m}{2}$, but we won't.

which edges in $S \cup T$ have already been added (and are thus part of Y) and which have been removed (and are thus part of X). Since we know that $C \cap Y = C \setminus X$, this is enough to reconstruct both $C \cap X$ and $C \cap Y$. (The same general idea works for paths, although for paths we do not need to be as careful to figure out orientation since we can only leave v_0 in one direction.)

The result is that given S and T, there are at most Nm^2 choices of X and Y such that the canonical path γ_{XY} crosses ST. This gives $\rho \leq \frac{Nm^2 N^{-2}}{N^{-1}\frac{1}{2m}} = 2m^3$, which gives $\tau_2 \leq 32m^6$. This is not great but it is at least polynomial in the size of the graph. For actual sampling, this translates to $O\Big(m^6\Big(\log N + \log \frac{1}{\epsilon}\Big)\Big) = O\Big(m^6\Big(m + \log \frac{1}{\epsilon}\Big)\Big)$ steps to get a total variation distance of ϵ.

This may or may not give rapid mixing, depending on how big N is relative to m. For many graphs, the number of matchings N will be exponential in m, and so the m^6 term will be polylogarithmic in N. But N can be much smaller in dense graphs. For example, on a clique we have $N = m + 1$, since no matching can contain more than one edge, making the bound $\tau_2 \leq 32m^6 = \Theta(N^6)$ polynomial in N but *not* polynomial in $\log N$, which is what we want. The actual mixing time in this case is $\Theta(m)$ (for the upper bound, wait to delete the only edge, and do so on both sides of the coupling; for the lower bound, observe that until we delete the only edge we are still in our initial state). This is much better than the upper bound from canonical paths, but it still doesn't give rapid maxing.

10.6.4.4 Perfect matchings in dense bipartite graphs

(Basically doing [MR95, §11.3].)

A similar chain can be used to sample perfect matchings in dense bipartite graphs, as shown by Jerrum and Sinclair [JS89] based on an algorithm by Broder [Bro86] that turned out to have a bug in the analysis [Bro88].

A **perfect matching** of a graph G is a subgraph that includes all the vertices and gives each of them exactly one incident edge. We'll be looking for perfect matchings in a **bipartite graph** consisting of n left vertices $u_1, \ldots, u_n$ and n right vertices $v_1, \ldots, v_n$, where every edge goes between a left vertex and a right vertex. We'll also assume that the graph is **dense**, which in this case means that every vertex has at least $n/2$ neighbors. This density assumption was used in the original Broder and Jerrum-Sinclair papers but removed in a later paper by Jerrum, Sinclair, and Vigoda [JSV04].

The random walk is similar to the random walk from the previous section restricted to the set of matchings with either $n - 1$ or n edges. At each step, we first flip a coin (the usual lazy walk trick); if it comes up heads,

we choose an edge uv uniformly at random and apply one of the following transformations depending on how uv fits in the current matching m_t:

1. If $uv \in m_t$, and $|m_t| = n$, set $m_{t+1} = m_t \setminus \{uv\}$.
2. If u and v are both unmatched in m_t, and $|m_t| = n - 1$, set $m_{t+1} = m_t \cup \{uv\}$.
3. If exactly one of u and v is matched to some other node w, and $|m_t| = n - 1$, perform a rotation that deletes the w edge and adds uv.
4. If none of these conditions hold, set $m_{t+1} = m_t$.

The walk can be started from any perfect matching, which can be found in $O(n^{5/2})$ time using a classic algorithm of Hopcroft and Karp [HK73] that repeatedly searches for an **augmenting path**, which is a path in G between two unmatched vertices that alternates between edges not in the matching and edges in the matching. (We'll see augmenting paths again below when we show that any near-perfect matching can be turned into a perfect matching using at most two transitions.)

We can show that this walk converges in polynomial time using a canonical path argument. This is done in two stages: first, we define canonical paths between all perfect matchings. Next, we define a short path from any matching of size $n - 1$ to some nearby perfect matching, and build paths between arbitrary matchings by pasting one of these short paths on one or both ends of a long path between perfect matchings. This gives a collection of canonical paths that we can show to have low congestion.

To go between perfect matchings X and Y, consider $X \cup Y$ as a collection of paths and even cycles as in §10.6.4.3. In some standard order, fix each path cycle by first deleting one edge to make room, then using rotate operations to move the rest of the edges, then putting the last edge back in. For any transition S–T along the way, we can use the same argument that we can compute XUY from SUT by supplying the missing edges, which will consist of a matching of size n plus at most one extra edge. So if N is the number of matchings of size n or $n - 1$, the same argument used previously shows that at most Nm of these long paths cross any S–T transition.

For the short paths, we must use the density property. The idea is that for any matching that is not perfect, we can find an augmenting path of length at most 3 between two unmatched nodes on either side. Pick some unmatched nodes u and v. Each of these nodes is adjacent to at least $n/2$ neighbors; if any of these neighbors are unmatched, we just found an augmenting path of length 1. Otherwise the $n/2$ neighbors of u and the $n/2$ nodes matched

to neighbors of v overlap (because v is unmatched, leaving at most $n-1$ matched nodes and thus at most $n/2-1$ nodes that are matched to something that's not a neighbor of v). So for each matching of size $n-1$, in at most two steps (rotate and then add an edge) we can reach some specific perfect matching. There are at most m^2 ways that we can undo this, so each perfect matching is associated with at most m^2 smaller matchings. This blows up the number of canonical paths crossing any transition by roughly m^4; by counting carefully we can thus show congestion that is $O(m^6)$ ($O(m^4)$ from the blow-up, m from the m in Nm, and m from $1/p_{ST}$).

It follows that for this process, $\tau_2 = O(m^{12})$. (I think a better analysis is possible.)

As noted earlier, this is an example of a process for which causal coupling doesn't work in less than exponential time [KR99], a common problem with Markov chains that don't have much symmetry. So it's not surprising that stronger techniques were developed specifically to attack this problem.

An issue we need to deal with is that when we sample a matching with this process, we don't necessarily get a perfect matching, since many states are going to have only $n-1$ edges. But we have already shown that the perfect matchings make up at least an m^{-2} fraction of the total. So if we don't get a perfect matching after a polynomial number of steps, we can simply start over and try again. This adds a polynomial factor to the already-bad running time, but it is still polynomial. Alternatively we can argue that the Markov chain sampled only at times when it yields a perfect matching is (a) still a Markov chain, and (b) has a total variation distance from its stationary distribution that we can bound using the bounds on the unsampled chain. The details of this are a little messy but it also works.

Chapter 11

Approximate counting

(See also [MR95, Chapter 11].)

The basic idea: we have some class of objects, and we want to know how many of them there are. Ideally we can build an algorithm that just prints out the exact number, but for many problems this is hard.

A **fully polynomial-time randomized approximation scheme** or **FPRAS** for a numerical problem outputs a number that is between $1 - \epsilon$ and $1 + \epsilon$ times the correct answer, with probability at least $3/4$ (or some constant bounded away from $1/2$—we can amplify to improve it), in time polynomial in the input size n and $1/\epsilon$. In this chapter, we'll be hunting for FPRASs. But first we will discuss briefly why we can't just count directly in many cases.

11.1 Exact counting

A typical application is to a problem in the complexity class #**P**, problems that involve counting the number of accepting computation paths in a nondeterministic polynomial-time Turing machine. Equivalently, these are problems that involve counting for some input x the number of values r such that $M(x, r) = 1$, where M is some machine in **P**. An example would be the problem **#SAT** of counting the number of satisfying assignments of some CNF formula.

The class #**P** (which is usually pronounced **sharp P** or **number P**) was defined by Leslie Valiant in a classic paper [Val79]. The central result in this paper is **Valiant's theorem**. This shows that any problem in #**P** can be **reduced** (by **Cook reductions**, meaning that we are allowed to use the target problem as a subroutine instead of just calling it once) to the

problem of computing the **permanent** of a square 0–1 matrix A, where the permanent is given by the formula $\sum_\pi \prod_i A_{i,\pi(i)}$, where the sum ranges over all $n!$ permutations π of the indices of the matrix. An equivalent problem is counting the number of **perfect matchings** (subgraphs including all vertices in which every vertex has degree exactly 1) of a bipartite graph. Other examples of #**P**-complete problems are #SAT (defined above) and **#DNF** (like #SAT, but the input is in DNF form; #SAT reduces to #DNF by negating the formula and then subtracting the result from 2^n).

Exact counting of #**P**-hard problems is likely to be very difficult: **Toda's theorem** [Tod91] says that being able to make even a single query to a #**P**-oracle is enough to solve any problem in the **polynomial-time hierarchy**, which contains most of the complexity classes you have probably heard of. Nonetheless, it is often possible to obtain good approximations to such problems.

11.2 Counting by sampling

If many of the things we are looking for are in the target set, we can count by sampling; this is what poll-takers do for a living. Let U be the universe we are sampling from and G be the "good" set of points we want to count. Let $\rho = |G|/|U|$. If we can sample uniformly from U, then we can estimate ρ by taking N independent samples and dividing the number of samples in G by N. This will give us an answer $\hat{\rho}$ whose expectation is ρ, but the accuracy may be off. Since the variance of each sample is $\rho(1-\rho) \approx \rho$ (when ρ is small), we get $\operatorname{Var}[\sum X_i] \approx m\rho$, giving a standard deviation of $\sqrt{m\rho}$. For this to be less than our allowable error $\epsilon m\rho$, we need $1 \leq \epsilon\sqrt{m\rho}$ or $N \geq \frac{1}{\epsilon^2\rho}$. This gets bad if ρ is exponentially small. So if we are to use sampling, we need to make sure that we only use it when ρ is large.

On the positive side, if ρ is large enough we can easily compute how many samples we need using Chernoff bounds. The following lemma gives a convenient estimate; it is based on [MR95, Theorem 11.1] with a slight improvement on the constant:

Lemma 11.2.1. *Sampling N times gives relative error ϵ with probability at least $1-\delta$ provided $\epsilon \leq 1.81$ and*

$$N \geq \frac{3}{\epsilon^2\rho} \ln \frac{2}{\delta}. \tag{11.2.1}$$

Proof. Suppose we take N samples, and let X be the total count for these

samples. Then $\mathrm{E}[X] = \rho N$, and (5.2.7) gives (for $\epsilon \leq 1.81$):

$$\Pr[|X - \rho N| \geq \epsilon \rho N] \leq 2e^{-\rho N \epsilon^2/3}.$$

Now set $2e^{-\rho N \epsilon^2/3} \leq \delta$ and solve for N to get (11.2.1). □

11.2.1 Generating samples

An issue that sometimes comes up in this process is that it is not always obvious how to generate a sample from a given universe U. What we want is a function that takes random bits as input and produces an element of U as output.

In the simplest case, we know $|U|$ and have a function f from $\{0, \ldots, |U| - 1\}$ to U (the application of such a function is called **unranking**). Here we can do **rejection sampling**: we choose a bit-vector of length $\lceil \lg |U| \rceil$, interpret it as an integer x expressed in binary, and try again until $x < |U|$. This requires less than $2\lceil \lg |U| \rceil$ bits on average in the worst case, since we get a good value at least half the time.

Unranking (and the converse operation of ranking) can be a non-trivial problem in combinatorics, and there are entire textbooks on the subject [SW13]. But some general principles apply. If U is a union of disjoint sets $U_1 \cup U_2 \cup \ldots \cup U_n$, we can compute the sizes of each $|U_i|$, and we can unrank each U_i, then we can map some $x \in \{0 \ldots |U| - 1\}$ to a specific element of U by choosing the maximum i such that $\sum_{j=1}^{i-1} |U_j| \leq x$ and choosing the k-th element of U_i, where $k = x - \sum_{j=1}^{i-1} |U_j|$. This is essentially what happens if we want to compute the x-th day of the year. The first step of this process requires computing the sizes of $O(n)$ sets if we generate the sum one element at a time, although if we have a fast way of computing $\sum_{i=1}^{j} |U_i|$ for each i, we can cut this time to $O(\log n)$ such computations down using binary search. The second step depends on whatever mechanism we use to unrank within U_i.

An example would be generating one of the $\binom{n}{k}$ subsets of a set of size k uniformly at random. If we let S be the set of all such k-subsets, we can express S as $\bigcup_{i=1}^{n} S_i$ where i is the set of all k-subsets that have the i-th element as their smallest element in some fixed ordering. But then we can easily compute $|S_i| = \binom{n-i}{k-1}$ for each i, apply the above technique to pick a particular S_i to start with, and then recurse within S_i to get the rest of the elements.

A more general case is when we can't easily sample from U but we can sample from some $T \supseteq U$. Here rejection sampling comes to our rescue

as long as $|U|/|T|$ is large enough. For example, if we want to generate a k-subset of n elements when $k \ll \sqrt{n}$, we can choose a list of k elements with replacement and discard any lists that contain duplicates. But this doesn't work so well for larger k. In this particular case, there is an easy out: we can sample k elements without replacement and forget their order. This corresponds to sampling from a universe T of ordered k-subsets, then mapping down to U. As long as the inverse image of each element of U has the same size in T, this will give us a uniform sample from U.

When rejection sampling fails, we may need to come up with a more clever approach to concentrate on the particular values we want. One approach that can work well if we can express U as a union of sets that are not necessarily disjoint is the Karp-Luby technique [KL85], discussed in §11.4 below.

All of this assumes that we are interested in getting uniform samples. For non-uniform samples, we replace the assumption that we are trying to generate a uniform $X \in \{0 \ldots m-1\}$ with some $X \in \{0 \ldots m-1\}$ for which we can efficiently calculate $\Pr[X \le i]$ for each i. In this case, we can (in principle) generate a continuous random variable $Y \in [0,1]$ and choose the maximum i such that $\Pr[X \le i] \le Y$, which we can find using binary search.

The complication is that we can't actually generate Y in finite time. Instead, we generate Y one bit at a time; this gives a sequence of rational values Y_t where each Y_t is of the form $k/2^t$. We can stop when every value in interval $[Y_t, Y_t + 2^{-t})$ lies with the range corresponding to some particular value of X. This technique is closely related to **arithmetic coding** [WNC87], and requires generating $H(X) + O(1)$ bits on average, where $H(X) = -\sum_{i=0}^{m-1} \Pr[X = i] \lg \Pr[X = i]$ is the (base 2) **entropy** of X.

11.3 Approximating #KNAPSACK

Here is an algorithm for approximating the number of solutions to a **KNAPSACK** problem, due to Dyer [Dye03]. We'll concentrate on the simplest version, 0–1 KNAPSACK, following the analysis in Section 2.1 of [Dye03].

For the 0–1 KNAPSACK problem, we are given a set of n objects of weight $0 \le a_1 \le a_2 \le \ldots a_n \le b$, and we want to find a 0–1 assignment $x_1, x_2, \ldots, x_n$ such that $\sum_{i=1}^{n} a_i x_i \le b$ (usually while optimizing some property that prevents us from setting all the x_i to zero). We'll assume that the a_i and b are all integers.

For #KNAPSACK, we want to compute $|S|$, where S is the set of all assignments to the x_i that make $\sum_{i=1}^{n} a_i x_i \le b$.

There is a well-known **fully polynomial-time approximation scheme** for optimizing KNAPSACK, based on dynamic programming. The idea is that a maximum-weight solution can be found exactly in time polynomial in b, and if b is too large, we can reduce it by rescaling all the a_i and b at the cost of a small amount of error. A similar idea is used in Dyer's algorithm: the KNAPSACK problem is rescaled so that size of the solution set S' of the rescaled version can be computed in polynomial time. Sampling is then used to determine what proportion of the solutions in S' correspond to solutions of the original problem.

Scaling step: Let $a'_i = \lfloor n^2 a_i / b \rfloor$. Then $0 \leq a'_i \leq n^2$ for all i. Taking the floor creates some error: if we try to reconstruct a_i from a'_i, the best we can do is argue that $a'_i \leq n^2 a_i / b < a'_i + 1$ implies $(b/n^2)a'_i \leq a_i < (b/n^2)a'_i + (b/n^2)$. The reason for using n^2 as our rescaled bound is that the total error in the upper bound on a_i, summed over all i, is bounded by $n(b/n^2) = b/n$, a fact that will become important soon.

Let $S' = \left\{ \vec{x} \mid \sum_{i=1}^{n} a'_i x_i \leq n^2 \right\}$ be the set of solutions to the rescaled knapsack problem, where we substitute a'_i for a_i and $n^2 = (n^2/b)b$ for b. Claim: $S \subseteq S'$. Proof: $\vec{x} \in S$ if and only if $\sum_{i=1}^{n} a_i x_i \leq b$. But then

$$\begin{aligned}
\sum_{i=1}^{n} a'_i x_i &= \sum_{i=1}^{n} \lfloor n^2 a_i / b \rfloor x_i \\
&\leq \sum_{i=1}^{n} (n^2/b) a_i x_i \\
&= (n^2/b) \sum_{i=1}^{n} a_i x_i \\
&\leq n^2,
\end{aligned}$$

which shows $\vec{x} \in S'$.

The converse does not hold. However, we can argue that any $\vec{x} \in S'$ can be shoehorned into S by setting at most one of the x_i to 0. Consider the set of all positions i such that $x_i = 1$ and $a_i > b/n$. If this set is empty, then $\sum_{i=1}^{n} a_i x_i \leq \sum_{i=1}^{n} b/n = b$, and $\vec{x}$ is already in S. Otherwise, pick any position i with $x_i = 1$ and $a_i > b/n$, and let $y_j = 0$ when $j = i$ and $y_j = x_j$ otherwise. If we recall that the total error from the floors in our approximation was at most $(b/n^2)n = b/n$, the intuition is that deleting y_j

removes it. Formally, we can write

$$\begin{aligned}\sum_{j=1}^{n} a_j y_j &= \sum_{j=1}^{n} a_j x_j - a_i \\ &< \sum_{j=1}^{n} ((b/n^2)a'_j + b/n^2)x_j - b/n \\ &\leq (b/n^2)\sum_{j=1}^{n} a'_j x_j + b/n - b/n \\ &\leq (b/n^2)n^2 \\ &= b.\end{aligned}$$

Applying this mapping to all elements $\vec{x}$ of S maps at most $n+1$ of them to each $\vec{y}$ in S'; it follows that $|S'| \leq (n+1)|S|$, which means that if we can sample elements of S' uniformly, each sample will hit S with probability at least $1/(n+1)$.

To compute $|S'|$, let $C(k,m) = \left|\left\{\vec{x} \;\middle|\; \sum_{i=1}^{k} a'_i x_i \leq m\right\}\right|$ be the number of subsets of $\{a'_1, \ldots, a'_k\}$ that sum to m or less. Then $C(k,m)$ satisfies the recurrence

$$\begin{aligned}C(k,m) &= C(k-1, m-a'_k) + C(k-1,m) \\ C(0,m) &= 1\end{aligned}$$

where k ranges from 0 to n and m ranges from 0 to n^2, and we treat $C(k-1, m-a'_k) = 0$ if $m - a'_k < 0$. The idea is that $C(k-1, m-a'_k)$ counts all the ways to make m if we include a'_k, and $C(k-1,m)$ counts all the ways to make m if we exclude it. The base case corresponds to the empty set (which sums to $\leq m$ no matter what m is).

We can compute a table of all values of $C(k,m)$ by iterating through m in increasing order; this takes $O(n^3)$ time. At the end of this process, we can read off $|S'| = C(n, n^2)$.

But we can do more than this: we can also use the table of counts to sample uniformly from S'. The probability that $x'_n = 1$ for a uniform random element of S' is exactly $C(n-1, n^2-a'_n)/C(n,n^2)$; having chosen $x'_n = 1$ (say), the probability that $x'_{n-1} = 1$ is then $C(n-2, n^2-a'_n-a'_{n-1})/C(n-2, n^2-a'_n)$, and so on. So after making $O(n)$ random choices (with $O(1)$ arithmetic operations for each choice to compute the probabilities) we get a uniform element of S', which we can test for membership in S in an additional $O(n)$ operations.

We've already established that $|S|/|S'| \geq 1/(n+1)$, so we can apply Lemma 11.2.1 to get ϵ relative error with probability at least $1-\delta$ using $\frac{3(n+1)}{\epsilon^2} \ln \frac{2}{\delta}$ samples. This gives a cost of $O(n^2 \log(1/\delta)/\epsilon^2)$ for the sampling step, or a total cost of $O(n^3 + n^2 \log(1/\delta)/\epsilon^2)$ after including the cost of building the table (in practice, the second term will dominate unless we are willing to accept $\epsilon = \omega(1/\sqrt{n})$).

It is possible to improve this bound. Dyer [Dye03] shows that using **randomized rounding** on the a'_i instead of just truncating them gives a FPRAS that runs in $O(n^{5/2}\sqrt{\log(1/\epsilon)} + n^2/\epsilon^2)$ time. We won't do this here, but we will see randomized rounding used in a different context in §13.2.2.

11.4 Approximating #DNF

A classical algorithm of Karp and Luby [KL85] gives a FPRAS for #DNF. We'll describe how this works, mostly following the presentation in [MR95, §11.2]. The key idea of the Karp-Luby technique is to express the set S whose size we want to know, as a union of a polynomial number of simpler sets $S_1, \ldots, S_k$. If for each i we can sample uniformly from S_i, compute $|S_i|$, and determine membership in S_i, then some clever sampling will let us approximate $|S|$ by sampling from a *disjoint* union of the S_i to determine how much $\sum |S_i|$ overestimates $|S|$.

To make this concrete, let's look at the specific problem studied by Karp and Luby of approximating the number of satisfying assignment to a DNF formula. A **DNF formula** is a formula that is in **disjunctive normal form**: it is an OR of zero or more **clauses**, each of which is an AND of variables or their negations. An example would be $(x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_4) \vee x_2)$. The **#DNF** problem is to count the number of satisfying assignments of a formula presented in disjunctive normal form.

Solving #DNF exactly is #**P**-complete, so we don't expect to be able to do it. Instead, we'll get a FPRAS by cleverly sampling solutions. The need for cleverness arises because just sampling solutions directly by generating one of the 2^n possible assignments to the n variables may find no satisfying assignments at all, since the size of any individual clause might be big enough that getting a satisfying assignment for that clause at random is exponentially unlikely.

So instead we will sample pairs (x, i), where x is an assignment that satisfies clause C_i; these are easier to find, because if we know which clause C_i we are trying to satisfy, we can read off the satisfying assignment from its variables. Let S be the set of such pairs. For each pair (x, i), define

$f(x,i) = 1$ if and only if $C_j(x) = 0$ for all $j < i$. Then $\sum_{(x,i)\in S} f(x,i)$ counts every satisfying assignment x, because (a) there exists some i such that x satisfies C_i, and (b) only the smallest such i will have $f(x,i) = 1$. In effect, f is picking out a single canonical satisfied clause from each satisfying assignment. Note that we can compute f efficiently by testing x against all clauses C_j with $j < i$.

Our goal is to estimate the proportion ρ of "good" pairs with $f(x,i) = 1$ out of all pairs in $S' = \biguplus S_i$, and then use this to estimate $|S| = \sum_{(x,i)\in S'} f(x,i) = \rho|S'|$. If we can sample from S uniformly, the proportion ρ of "good" pairs with $f(x,i) = 1$ is at least $1/m$, because every satisfying assignment x contributes at most m pairs total, and one of them is good.

The only tricky part is figuring out how to sample pairs (x,i) with $C_i(x) = 1$ so that all pairs occur with the same probability. Let $S_i = \{(x,i) \mid C_i(x) = 1\}$. Then we can compute $|S_i| = 2^{n-k_i}$ where k_i is the number of literals in C_i. Using this information, we can sample i first with probability $|S_i|/\sum_j |S_j|$, then sample x from S_i just by picking independent uniform random values for the $n-k$ variables not fixed by C_i.

With $N \geq \frac{4}{\epsilon^2(1/m)} \ln \frac{2}{\delta} = \frac{4m}{\epsilon^2} \ln \frac{2}{\delta}$, we obtain an estimate $\hat{\rho}$ for the proportion of pairs (x,i) with $f(x,i) = 1$ that is within ϵ relative error of ρ with probability at least $1-\delta$. Multiplying this by $\sum |S_i|$ then gives the desired count of satisfying assignments.

It's worth noting that there's nothing special about DNF formulas in this method. Essentially the same trick will work for estimating the size of the union of any collection of sets S_i where we can (a) compute the size of each S_i; (b) sample from each S_i individually; and (c) test membership of our sample x in S_j for $j < i$.

11.5 Approximating exponentially improbable events

For #KNAPSACK and #DNF, we saw how restricting our sampling to a cleverly chosen sample space could boost the hit ratio ρ to something that gave a reasonable number of samples using Lemma 11.2.1. For other problems, it is often not clear how to do this directly, and the best sample spaces we can come up with make our target points an exponentially small fraction of the whole.

In these cases, it is sometimes possible to approximate this exponentially small fraction as a product of many more reasonable ratios. The idea is to express our target set as the last of a sequence of sets $S_0, S_1, \ldots, S_k$, where we can compute the size of S_0 and can estimate $|S_{i+1}|/|S_i|$ accurately for

each i. This gives $|S_k| = |S_0| \cdot \prod_{i=1}^{k} \frac{|S_{i+1}|}{|S_i|}$, with a relative error that grows roughly linearly with k. Specifically, if we can approximate each $|S_{i+1}|/|S_i|$ ratio to between $1-\epsilon$ and $1+\epsilon$ of the correct value, then the product of these ratios will be between $(1-\epsilon)^k$ and $(1+\epsilon)^k$ of the correct value; these bounds approach $1-\epsilon k$ and $1+\epsilon k$ in the limit as ϵk goes to zero, using the binomial theorem, although to get a real bound we will need to do more careful error analysis.

11.5.1 Matchings

We saw in §10.6.4.3 that a random walk on matchings on a graph with m edges has mixing time $\tau_2 \leq 32m^6$, where the walk is defined by selecting an edge uniformly at random and flipping whether it is in the matching or not, while rejecting any steps that produce a non-matching. This allows us to sample matchings of a graph with δ total variation distance from the uniform distribution in $O\left(m^6\left(\log N + \log \frac{1}{\delta}\right)\right)$ time, where N is the number of matchings. Since every matching is a subset of the edges, we can crudely bound $N \leq 2^m$ which lets us rewrite the sampling time as $O\left(m^6\left(m + \log \frac{1}{\delta}\right)\right)$.

Suppose now that we want to count matchings instead of sampling them. It's easy to show that for any particular edge $uv \in G$, at least half of all matchings in G don't include uv: the reason is that if M is a matching in G, then $M' = M \setminus \{uv\}$ is also a matching, and at most two matchings M' and $M' \cup \{uv\}$ are mapped to any one M' by this mapping.

Order the edges of G arbitrarily as $e_1, e_2, \ldots, e_m$. Let S_i be the set of matchings in $G \setminus \{e_1 \ldots e_i\}$. Then S_0 is the set of all matchings, and we've just argued that $\rho_{i+1} = |S_{i+1}|/|S_i| \geq 1/2$. We also know that $|S_m|$ counts the number of matchings in a graph with no edges, so it's exactly one. So we can use the product-of-ratios trick to compute $S_0 = \prod_{i=1}^{m} \frac{|S_i|}{|S_{i+1}|}$.

A random walk of length $O\left(m^6\left(m + \log \frac{1}{\eta}\right)\right)$ can sample matchings from S_i with a probability ρ' of getting a matching in S_{i+1} that is between $(1-\eta)\rho_{i+1}$ and $(1+\eta)\rho_{i+1}$. From Lemma 11.2.1, we can estimate ρ' within relative error γ with probability at least $1-\zeta$ using $O\left(\frac{1}{\gamma^2\rho'} \log \frac{1}{\zeta}\right) = O\left(\frac{1}{\gamma} \log \frac{1}{\zeta}\right)$ samples. Combined with the error on ρ', this gives relative error at most $\gamma + \eta + \gamma\eta$ in $O\left(m^6\left(m + \log \frac{1}{\eta}\right) \log \frac{1}{\gamma} \log \frac{1}{\zeta}\right)$ operations.[1] If we then multiply

[1] This is the point where sensible people start hauling out the $\widetilde{O}$ notation, where a function is $\widetilde{O}(f(n))$ if it $O(f(n)g)$ where g is polylogarithmic in n and any other parameters that may be running around ($\frac{1}{\epsilon}$, $\frac{1}{\eta}$, etc.).

out all the estimates for $|S_i|/|S_{i+1}|$, we get an estimate of S_0 that is at most $(1+\gamma+\eta+\gamma\eta)^m$ times the correct value with probability at least $1-m\zeta$ (with a similar bound on the other side), in total time $O\left(m^7\left(m \log \frac{1}{\eta}\right) \log \frac{1}{\gamma} \log \frac{1}{\zeta}\right)$.

To turn this into a fully polynomial-time approximation scheme, given ϵ, δ, and m, we need to select η, γ, and ζ to get relative error ϵ with probability at least $1-\delta$. Letting $\zeta = \delta/m$ gets the δ part. For ϵ, we need $(1+\gamma+\eta+\gamma\eta)^m \leq 1+\epsilon$. Suppose that $\epsilon < 1$ and let $\gamma = \eta = \epsilon/6m$. Then

$$\begin{aligned}(1+\gamma+\eta+\gamma\eta)^m &\leq \left(1+\frac{\epsilon}{2m}\right)^m \\ &\leq e^{\epsilon/2} \\ &\leq 1+\epsilon.\end{aligned}$$

Plugging these values into our cost formula gives $O(m^8)$ times a bunch of factors that are polynomial in $\log m$ and $\frac{1}{\epsilon}$, which we can abbreviate as $\widetilde{O}(m^8)$.

11.5.2 Other problems

Similar methods work for other problems that self-reduce by restricting particular features of the solution. Examples include colorings (fix the color of some vertex), independent sets (remove a vertex), and approximating the volume of a convex body (take the intersection with a sphere of appropriate radius; see [MR95, §11.4] for a slightly less sketchy description). We will not go into details on any of these applications here.

Chapter 12

Hitting times

In addition to using Markov chains for sampling, we can also use Markov chains to model processes that we hope will eventually reach some terminating state. These Markov chains will generally not be irreducible, since often the terminating states are inescapable, and may or may not have the other desirable properties we needed for convergence analysis. So instead of looking at convergence to a stationary distribution that might not even exist, we will be mostly interested in the **hitting time** for some subset of A of the states, defined as the minimum τ such that $X^\tau \in A$ starting from some given initial distribution on X^0. As suggested by the notation, hitting times will a special case of stopping times (see Chapter 9), and if we are very lucky we may be able to use the optional stopping theorem to bound them. If we are less lucky, we will use whatever tools we can.

12.1 Waiting times

The simplest case is when our Markov chain consists of two states: one where something hasn't happened yet, and one where it has. If the something happens with probability p at each step, then τ has a geometric distribution, and we can calculate $\mathrm{E}[\tau]$ using the usual conditional-probability argument for a geometric random variable (see §3.6.2):

$$\begin{aligned}\mathrm{E}[\tau] &= p \cdot 1 + (1-p) \cdot (1 + \mathrm{E}[\tau]) \\ &= 1 + (1-p)\,\mathrm{E}[\tau]\end{aligned}$$

which has the solution

$$\mathrm{E}[\tau] = 1/p.$$

A more interesting case is when we have a Markov chain that we can describe using a sequence of waiting times. For example, an **epidemic** process involves a population of n agents of which initially only one is infected, and at each step we pair two agents chosen uniformly at random, and if one is infected and the other not, the uninfected agent becomes infected. Such processes are used as the basis for **gossip** or **epidemic algorithms** [DGH+87], where the infection corresponds to knowing some piece of information we want to broadcast through the population.

If we let X^t be the number of agents infected after t steps, we have a Markov process where

$$\begin{aligned} p_{k,k+1} &= \frac{k(n-k)}{\binom{n}{2}} \\ p_{k,k} &= 1 - p_{k,k+1}, \end{aligned}$$

and all other transition probabilities are 0. So any trajectory of this process involves starting in 1, moving to 2 after some time, then 3, and so on until all n agents are infected.

The hitting time τ_n for n will be the sum of the waiting times for each of these steps. This gives

$$\begin{aligned} \mathrm{E}\left[\tau_n\right] &= \sum_{k=1}^{n-1} \frac{1}{p_{k,k+1}} \\ &= \sum_{k=1}^{n-1} \frac{\binom{n}{2}}{k(n-k)} \\ &= \binom{n}{2} \sum_{k=1}^{n-1} \left(\frac{1/n}{k} + \frac{1/n}{n-k} \right) \\ &= \frac{(n-1)}{2} \sum_{k=1}^{n-1} \left(\frac{1}{k} + \frac{1}{n-k} \right) \\ &= (n-1) H_{n-1}. \end{aligned}$$

12.2 Lyapunov functions

Waiting time analysis works well for processes like the epidemic in §12.1 that only go in one direction. But what if sometimes we go backwards, or if it's not even possible to arrange the states of our Markov chain neatly in one dimension? Here it can be helpful to construct a **Lyapunov function**,

which is often called a **potential function** in the computer science literature. This is a function that reduces each state of a complex process to a single value in $\mathbb{R}$, ideally producing some sort of semimartingale that drops in a predictable way that so that we can use it to get a bound on the time to reach our target set.

We've seen examples of this already in §9.4.1. For an unbiased random walk, we saw that the function X_t^2 drops by 1 on average, making $X_t^2 - t$ a martingale. A similar result holds with $X_t - (p-q)t$ for biased random walks. In both cases, we replace the original process with a new process whose behavior involves time as a term.

Let's try this for a more complicated process. The **rock-paper-scissors** process is similar to the epidemic process, except that instead of having two states infected and uninfected, where infected overwrites uninfected when agents in these states meet, we have three states with a cyclic dominance relation. This means that when a rock agent meets a paper agent, we get two paper agents ("paper covers rock"), and similarly paper gets replaced by scissors ("scissors cuts paper") and scissors gets replaced by rock ("rock breaks scissors"). Meetings between agents in the same state have no effect. As in the epidemic case, we choose pairs of agents to interact uniformly at random from some population of size n.[1]

Processes like this have been extensively studied in a variety of models; see [SMJ+14] for a survey. The rock-paper-scissors process in particular is often modeled in its continuous limit as a system of differential equations:

$$\begin{aligned} r' &= rs - pr \\ p' &= pr - sp \\ s' &= sp - rs, \end{aligned}$$

where r, p, and s represent the densities of rocks, papers, and scissors in the population at some time and primes are used to indicate derivatives with respect to time.

This system of differential equations does not have simple closed-form solutions, but it does have some interesting invariants. If we compute $(r+p+s)' = r' + p' + s' = (rs - pr) + (pr - sp) + (sp - rs) = 0$, we see that the total concentration of agents $r + p + s$ doesn't change over time. This is

[1] This is also a special case of the **Lotka-Volterra model** in population dynamics. The translation is to convert rocks to wolves, paper to grass, and scissors to sheep, and adopt the rules "sheep eats grass", "wolf eats sheep", and "grass starves wolf."

not especially surprising. But we can also compute

$$\begin{aligned}(rps)' &= r'ps + rp's + rps' \\ &= (rs - pr)ps + rs(pr - sp) + rp(sp - rs) \\ &= rps(s - p) + rps(r - s) + rps(p - r) \\ &= 0.\end{aligned}$$

This shows that the product of the three concentrations also doesn't change over time, giving a family of closed-loop trajectories in the continuous case. This makes the rock-paper-scissors process a useful way to build an oscillator in a dynamical system.

In the discrete case, we don't have infinitesimal rocks breaking infinitesimal scissors over infinitesimal time intervals. Instead, we have a Markov chain. The states of this Markov chain are triples of counts $\langle R_t, P_t, S_t \rangle$, and at each step we see transitions

$$\begin{aligned}\langle R, P, S \rangle &\to \langle R - 1, P + 1, S \rangle && \text{with probability } RP/\binom{n}{2} \\ \langle R, P, S \rangle &\to \langle R, P - 1, S + 1 \rangle && \text{with probability } PS/\binom{n}{2} \\ \langle R, P, S \rangle &\to \langle R + 1, P, S - 1 \rangle && \text{with probability } SR/\binom{n}{2},\end{aligned}$$

with the remaining probability causing no change.

This is a finite Markov chain that is aperiodic because of the no-op transitions, but it's not irreducible. The three states $\langle n, 0, 0 \rangle$, $\langle 0, n, 0 \rangle$, and $\langle 0, 0, n \rangle$ all have no outgoing transitions: once we're all rocks, we all stay rocks forever. Furthermore there is a path from every configuration with n agents that isn't one of these three terminal configurations to one of these configurations. This means that eventually this process will converge to a uniform configuration, which is trouble if we want to use the process as an oscillator but maybe not so bad if we are trying to achieve agreement. But whatever our goal is, we can ask how long will this take?

The *rps* invariant for the continuous process suggests looking for a similar invariant or near-invariant for the discrete process. Let's define a Lyapunov function

$$\Phi = RPS,$$

and look at the expected change in this function at each step.

Suppose paper covers rock, which occurs with probability $PR/\binom{n}{2}$. Then the change in Φ is

$$\begin{aligned}\Delta\Phi &= (R-1)(P+1)S - RPS \\ &= RPS + RS - PS - S - RPS \\ &= (R-P-1)S.\end{aligned}$$

If we multiply this by the probability that the event occurs, we get a contribution to $\mathrm{E}[\Delta\Phi]$ of $(R-P-1)\cdot RPS/\binom{n}{2}$. By symmetry, the other two transitions yield contributions of $(P-S-1)\cdot RPS/\binom{n}{2}$ and $(S-R-1)\cdot RPS/\binom{n}{2}$. Adding these up gives

$$\mathrm{E}[\Delta\Phi] = \frac{RPS}{\binom{n}{2}}((R-P-1)+(P-S-1)+(S-R-1)),$$

which very nicely cancels down to

$$= RPS\cdot\frac{-3}{\binom{n}{2}}.$$

What this tells us is that Φ drops, on average, by a $\Theta(n^{-2})$ fraction of its current value. We can express this as

$$\mathrm{E}[\Phi_{t+1} \mid \Phi_t] = \Phi_t\left(1-\frac{3}{\binom{n}{2}}\right),$$

which, iterated, gives for any t

$$\mathrm{E}[\Phi_t] = \Phi_0\left(1-\frac{3}{\binom{n}{2}}\right)^t. \tag{12.2.1}$$

Alternatively we can use this to argue that

$$Z_t = \Phi_t\left(1-\frac{3}{\binom{n}{2}}\right)^{-t} \tag{12.2.2}$$

is a martingale.

For proving an upper bound on the hitting time, the first formulation (12.2.1) is more useful. Using our old friend $1+x\le e^x$, we can rewrite the

bound as $\mathrm{E}[\Phi_t] \le e^{-6t/n(n-1)}$. Now observe that Φ is never greater than $(n/3)^3$ and never less than $n-2$ as long as it is not zero. So we can compute

$$\begin{aligned}\Pr[\Phi_t \neq 0] &= \Pr[\Phi_t \ge n-2] \\ &\le \frac{\mathrm{E}[\Phi_t]}{n-2} \\ &\le \frac{(n/3)^3 e^{-6t/n(n-1)}}{n-2} \\ &= O\Big(n^2 \exp\Big(-\Theta(t/n^2)\Big)\Big).\end{aligned}$$

We can immediately see that there is some value $t = O(n^2 \log n)$ that knocks $\Pr[\Phi_t \neq 0]$ down to at most $1/2$. As when converging to a stationary distribution, if we lose this coin-flip, we can restart the argument and try again. This gives an expected waiting time of $O(n^2 \log n)$ to reach $\Phi = 0$.

We are not quite done. Having $\Phi = 0$ only means that one of our three species has disappeared; for full convergence, we need to lose two. But once we are down to two remaining species (say rock and paper), we have an epidemic process. From our previous analysis, we know that only $O(n \log n)$ additional steps are needed to get down to one.

It's worth noting that hitting times are generally stopping times. So perhaps there is a cleaner argument using (12.2.2) and the optional stopping theorem (see Theorem 9.3.1). Let τ be the first time at which $\Phi_\tau = 0$. We've already shown that $\mathrm{E}[\tau]$ is finite, so the bounded-increments case of Theorem 9.3.1 is tempting. But if we get unlucky and $\Phi_{t+1} = \Phi_t \neq 0$ for some large t, Z_t can increase by an arbitrarily large amount. The bounded time and bounded range cases also don't apply. It's tempting to see if the general case works, but since we know that $Z_\tau = 0 \neq Z_0$ and OST implies $Z_\tau = Z_0$, we can rule out applying the theorem to this particular martingale and stopping time no matter how clever we get. What we *can* do is use a fixed time t and compute $\mathrm{E}[Z_t] = Z_0 = \Phi_0$, but at this point we are just reinventing (12.2.1).

To salve our disappointment, let's get a lower bound on the expected fixation time. Use $1 - x \ge e^{-x/2}$, which holds for $0 \le x \le 2$ at least, to get

$$\begin{aligned}\mathrm{E}[\Phi_t] &= \Phi_0\Big(1 - \Theta(1/n^2)\Big)^t \\ &\ge \Phi_0 \exp\Big(-\Theta(t/n^2)\Big).\end{aligned}$$

If we start with $\Phi_0 = \Theta(n^3)$, there is some $t = \Theta(n^2)$ such that $\mathrm{E}[\Phi_t]$ is

still $\Theta(n^3)$, just with a smaller constant. But then we can argue

$$\begin{aligned}\Theta(n^3) &= \mathrm{E}\left[\Phi_t\right] \\ &= \mathrm{E}\left[\Phi_t\right] \Pr\left[\Phi_t \neq 0\right] \\ &\leq O(n^3) \Pr\left[\Phi_t \neq 0\right],\end{aligned}$$

from which $\Pr\left[\Phi_t \neq 0\right] = \Omega(1)$. So there is a constant probability of not reaching $\Phi_t = 0$ after $\Theta(n^2)$ steps, which gives an expected hitting time for $\Phi = 0$ of $\Omega(n^2)$. This bound is missing a log factor compared to our upper bound. I suspect that this could be improved with a better analysis.

12.3 Drift analysis

The Lyapunov/potential function approach has two main steps: (a) finding a good Lyapunov function and (b) using it to get a bound on hitting time. The first step can be frustrating because we know that a good Lyapunov function always exists, since we can in principle just take the expected hitting time from the current state. But finding a good, clean Lyapunov function that we can write down and reason about is tougher. The second step can involve a lot of awkward futzing with supermartingales and such, and we can avoid some of this awkwardness but letting other people package it up in a few convenient theorems. For this purpose, it's helpful to look at the literature on **drift analysis**, which mostly shows up in the context of **evolutionary algorithms**, a class of optimization algorithms modeled on natural evolutionary processes. In this section, we'll describe some of the results in this area, largely following a survey by Lengler [Len20]. Many of these results are at their core repackagings of the optional stopping theorem, but it can be convenient not to have to go through the details of the OST every time we want to use them.

As before, the idea is that we will come up with some one-dimensional stochastic process $\{X_t\}$ on the non-negative reals $\mathbb{R}_0^+$ that is related to the process we are trying to bound. Our hope is that X_t either drops by a consistent additive amount or a consistent multiplicative factor on average at each step.

For **additive drift**, we have the following:

Theorem 12.3.1 ([Len20, Theorem 2.3.1], adapted from [HY04])**.** *Let $(X_t)_{t\geq 0}$ b e sequence of non-negative random variables with a finite state space $S \subseteq \mathbb{R}_0^+$ such that $0 \in S$. Let $T = \inf\{t \geq 0 \mid X_t = 0\}$.*

1. *If there exists* $\delta > 0$ *such that for all* $s \in S \setminus \{0\}$ *and for all* $t \geq 0$,

$$\Delta_t(s) = \mathrm{E}\left[X_t - X_{t+1} \mid X_t = s\right] \geq \delta,$$

then

$$\mathrm{E}\left[T\right] \leq \frac{\mathrm{E}\left[X_0\right]}{\delta}.$$

2. *If there exists* $\delta > 0$ *such that for all* $s \in S \setminus \{0\}$ *and for all* $t \geq 0$,

$$\Delta_t(s) = \mathrm{E}\left[X_t - X_{t+1} \mid X_t = s\right] \leq \delta,$$

then

$$\mathrm{E}\left[T\right] \geq \frac{\mathrm{E}\left[X_0\right]}{\delta}.$$

The proof of Theorem 12.3.1 is a fairly simple application of the optional stopping theorem. Depending on which case we are in, $X_t + t\delta$ is either a supermartingale or submartingale; we have a bounded range because of the assumption that S is finite; and we have finite time in the supermartingale case because the downward drift implies and finite state space implies that there is always a path to 0 that we can take with nonzero probability, and in the submartingale case because if we don't $\mathrm{E}\left[T\right]$ is infinite. So in either case we are looking at arguments that we've done before.

Where the theorem is helpful is that it saves us from having to repeat these arguments. For example, in the case of a biased random walk with reflecting barriers at 0 and n and a probability of $p > 1/2$ of dropping at each step, we can compute $\mathrm{E}\left[X_t - X_{t+1} \mid X_t = s\right] \geq p - q$ (note that the convention here is that we are tracking expected drop, which is the negative of the expected change) and instantly get $\mathrm{E}\left[T\right] \leq n/(p-q)$ for any starting point $X_0 \leq n$. In the other direction we are not so lucky: the reflecting barrier at n means that $\mathrm{E}\left[X_t - X_{t+1} \mid X_t = n\right] = 1$, so the best bound we get starting at $X_0 = n$ is $\mathrm{E}\left[T\right] \geq n$, which we don't really need the theorem to get.

In some cases, it can be hard to find a Lyapunov function with constant additive drift. Rescaling may help in this case: the idea is to stretch the Lyapunov function locally by multiplying by the inverse of the expected drop, so that the new stretched function has expected drop at least 1. This trick has been reinvented several times in different contexts, dating back at least to the **probabilistic recurrence relations** of Karp *et al.* [KUW88]. We'll quote the version from the evolutionary analysis literature given by Lengler [Len20], which is known as the **variable drift** theorem:

Theorem 12.3.2 ([Len20, Theorem 2.3.3], adapted from [Joh10, RS14])**.** *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables with a finite state space $S \subseteq \mathbb{R}_0^+$ such that $0 \in S$. Let $s_{\min} = \min(S \setminus \{0\})$, let $T = \inf\{t \geq 0 \mid X_t = 0\}$, and for $t \geq 0$ and $s \in S$ let $\Delta_t(s) = \mathrm{E}[X_t - X_{t+1} \mid X_t = s]$. If there is an increasing function $h : \mathbb{R}^+ \to \mathbb{R}^+$ such that for all $s \in S \setminus \{0\}$ and all $t \geq 0$,*

$$\Delta_t(s) \geq h(s),$$

then

$$\mathrm{E}[T] \leq \frac{s_{\min}}{h(s_{\min})} + \mathrm{E}\left[\int_{s_{\min}}^{X_0} \frac{1}{h(\sigma)} d\sigma\right],$$

where the expectation in the latter term is over the random choice of X_0.

For the full proof, see [Len20, Theorem 2.3.3] or the original [Joh10, Theorem 4.6]. The intuition (using the notation of [Len20]) is that we can apply Theorem 12.3.1 to a function

$$g(s) = \begin{cases} \frac{s_{\min}}{h(s_{\min})} + \int_{s_{\min}}^{s} \frac{1}{h(\sigma)} d\sigma & \text{when } s \geq s_{\min}, \\ \frac{s}{s_{\min}} & \text{when } s \leq s_{\min}. \end{cases}$$

The slope of this function for large s is $1/h(s)$, which means that the expected change in $g(s)$ when we drop by $h(s)$ is roughly $h(s) \cdot (1/h(s)) = 1$. The exact bound requires observing that the integral is concave and applying Jensen's inequality. For s close to $s_{\min}$, the linear term covers any drop that goes below $s_{\min}$ (where we don't care about $h(s)$, because we are never going to land there).

The actual proof does a case analysis on all possible changes in s depending on whether they involve going up, going down but staying about $s_{\min}$, or going down and crossing to 0. Adding up all of these cases shows an expected drift for $g(s)$ of at least 1, reducing to Theorem 12.3.1. The somewhat messy bound is just what we get when we run the bound from the theorem back through g.

A special case of variable drift is **multiplicative drift**, where $\Delta_t(s) \geq \delta s$ for some constant $\delta > 0$. In this case the bound in Theorem 12.3.2 simplifies to

$$\mathrm{E}[T] \leq \frac{1 + \mathrm{E}[\ln(X_0/s_{\min})]}{\delta}. \tag{12.3.1}$$

For example, we showed in §12.2 that the function RPS for the rock-paper-scissors process has multiplicative drift with $\delta = 3/\binom{n}{2}$, $s_{\min} = n - 2$,

and $X_0 \leq (n/3)^3$. Plugging these values in (12.3.1) gives a bound on the time to reach $RPS = 0$ of $\frac{1+\ln((n/3)^3/(n-2))}{\binom{n}{2}} \approx \frac{3}{2}n^2 \ln n$. So we can get the same bound as before without having to go through all the intermediate steps.

There are many drift theorem results in the literature, including lower bounds and tail inequalities. The Lengler survey [Len20] summarizes many of these results. Another good source for Lyapunov-style bounds in general is the textbook of Menshikov *et al.* [MPW16].

Chapter 13

The probabilistic method

The **probabilistic method** is a tool for proving the existence of objects with particular combinatorial properties, by showing that some process generates these objects with nonzero probability.

The relevance of this to randomized algorithms is that in some cases we can make the probability large enough that we can actual produce such objects.

We'll mostly be following Chapter 5 of [MR95] with some updates for more recent results. If you'd like to read more about these technique, a classic reference on the probabilistic method in combinatorics is the text of Alon and Spencer [AS92].

13.1 Randomized constructions and existence proofs

Suppose we want to show that some object exists, but we don't know how to construct it explicitly. One way to do this is to devise some random process for generating objects, and show that the probability that it generates the object we want is greater than zero. This implies that something we want exists, because otherwise it would be impossible to generate; and it works even if the nonzero probability is very, very small. The systematic development of the method is generally attributed to the notoriously productive mathematician Paul Erdős and his frequent collaborator Alfréd Rényi.

From an algorithmic perspective, the probabilistic method is useful mainly when we can make the nonzero probability substantially larger than zero—and especially if we can recognize when we've won. But sometimes just demonstrating the existence of an object is a start.

We give a couple of example of the probabilistic method in action below.

In each case, the probability that we get a good outcome is actually pretty high, so we could in principle generate a good outcome by retrying our random process until it works. There are some more complicated examples of the method for which this doesn't work, either because the probability of success is vanishingly small, or because we can't efficiently test whether what we did succeeded (the last example below may fall into this category). This means that we often end up with objects whose existence we can demonstrate even though we can't actually point to any examples of them. For example, it is known that there exist **sorting networks** (a special class of circuits for sorting numbers in parallel) that sort in time $O(\log n)$, where n is the number of values being sorted [AKS83], and these these can be generated randomly with nonzero probability. But the best explicit constructions of such networks take time $\Theta(\log^2 n)$, and the question of how to find an *explicit* network that achieves $O(\log n)$ time has been open for decades despite many efforts to solve it.

13.1.1 Set balancing

Here we have a collection of vectors $v_1, v_2, \ldots, v_n$ in $\{0,1\}^m$. We'd like to find ± 1 coefficients $\epsilon_1, \epsilon_2, \ldots \epsilon_n$ that minimize the **discrepancy** $\max_j |X_j|$ where $X_j = \sum_{i=1}^n \epsilon_i v_{ij}$.

This is called **set balancing** because we are trying to balance attributes (represented by the vectors) between two sets (represented by the coefficients). We can also take the transpose and imagine that we have a collection of sets (represented by the columns of the matrix made up of the v_i) that we are trying to divide in half as evenly as possible by assigning their common elements to the -1 and $+1$ piles.

If we choose the ϵ_i randomly, Hoeffding's inequality (5.3.2) says for each fixed j that $\Pr[|X_j| > t] < 2\exp(-t^2/2n)$ (since there are at most n non-zero values v_{ij}). Setting $2\exp(-t^2/2n) \leq 1/m$ gives $t \geq \sqrt{2n \ln 2m}$. So by the union bound, we have $\Pr\left[\max_j |X_j| > \sqrt{2n \ln 2m}\right] < 1$: a solution exists.

Since we only proved that the probability of winning in nonzero, this doesn't necessary give us a good solution. In §14.4.3 we will revisit this problem and show how to remove the randomness while still getting the same bound.

13.1.2 Ramsey numbers

Consider a collection of n schoolchildren, and imagine that each pair of schoolchildren either like each other or dislike each other. We assume that

these preferences are symmetric: if x likes y, then y likes x, and similarly if x dislikes y, y dislikes x. Let $R(k,h)$ be the smallest value for n that ensures that among any group of n schoolchildren, there is either a subset of k children that all like each other or a subset of h children that all dislike each other.[1]

It is not hard to show that $R(k,h)$ is finite for all k and h.[2] The exact value of $R(k,h)$ is known only for small values of k and h.[3] But we can use the probabilistic method to show that for $k = h$, it is reasonably large. The following theorem is due to Erdős, and was the first known lower bound on $R(k,k)$.

Theorem 13.1.1 ([Erd47])**.** *If $k \geq 3$, then $R(k,k) > 2^{k/2}$.*

Proof. Suppose each pair of schoolchildren flip a fair coin to decide whether they like each other or not. Then the probability that any particular set of k schoolchildren all like each other is $2^{-\binom{k}{2}}$ and the probability that they all dislike each other is the same. Summing over both possibilities and all subsets gives a bound of $\binom{n}{k}2^{1-\binom{k}{2}}$ on the probability that there is at least one subset in which everybody likes everybody or everybody dislikes everybody.

[1] In terms of graphs, any graph G with at least $R(k,h)$ nodes contains either a **clique** of size k or an **independent set** of size h.

[2] A simple proof, due to Erdős and Szekeres [ES35], proceeds by showing that $R(k,h) \leq R(k-1,h) + R(k,h-1)$. Given a graph G of size at least $R(k-1,h) + R(k,h-1)$, choose a vertex v and partition the graph into two induced subgraphs G_1 and G_2, where G_1 contains all the vertices adjacent to v and G_2 contains all the vertices not adjacent to v. Either $|G_1| \geq R(k-1,h)$ or $|G_2| \geq R(k,h-1)$. If $|G_1| \geq R(k-1,h)$, then G_1 contains either a clique of size $k-1$ (which makes a clique of size k in G when we add v to it) or an independent set of size h (which is also in G). Alternatively, if $|G_2| \geq R(k,h-1)$, then G_2 either gives us a clique of size k by itself or an independent set of size h after adding v. Together with the fact that $R(1,h) = R(k,1) = 1$, this recurrence gives $R(k,h) \leq \binom{(k-1)+(h-1)}{k-1}$.

[3] There is a fairly current table at http://en.wikipedia.org/wiki/Ramsey's_Theorem. Some noteworthy values are $R(3,3) = 6$, $R(4,4) = 18$, and $43 \leq R(5,5) \leq 48$. One problem with computing exact values is that as $R(k,h)$ grows, the number of graphs one needs to consider gets very big. There are $2^{\binom{n}{2}}$ graphs with n vertices, and even detecting the presence or absence of a moderately-large clique or independent set in such a graph can be expensive. This pretty much rules out any sort of brute-force approach based on simply enumerating candidate graphs. But even smarter approaches have yielded only slow progress: for example, the upper bound on $R(5,5)$ was only improved from 49 [MR97] to 48 [AM18] after roughly two decades of work.

For $n = 2^{k/2}$, we have

$$\begin{aligned}\binom{n}{k}2^{1-\binom{k}{2}} &\leq \frac{n^k}{k!}2^{1-\binom{k}{2}} \\ &= \frac{2^{k^2/2+1-k(k-1)/2}}{k!} \\ &= \frac{2^{k^2/2+1-k^2/2+k/2}}{k!} \\ &= \frac{2^{1+k/2}}{k!} \\ &< 1.\end{aligned}$$

Because the probability that there is an all-liking or all-hating subset is less than 1, there must be some chance that we get a collection that doesn't have one. So such a collection exists. It follows that $R(k,k) > 2^{k/2}$, because we have shown that not all collections at $n = 2^{k/2}$ have the Ramsey property. □

The last step in the proof uses the fact that $2^{1+k/2} < k!$ for $k \geq 3$, which can be tested explicitly for $k = 3$ and proved by induction for larger k. The resulting bound is a little bit weaker than just saying that n must be large enough that $\binom{n}{k}2^{1-\binom{k}{2}} \geq 1$, but it's easier to use.

The proof can be generalized to the case where $k \neq h$ by tweaking the bounds and probabilities appropriately. Note that even though this process generates a graph with no large cliques or independent sets with reasonably high probability, we don't have any good way of testing the result, since testing for the existence of a clique is **NP**-hard.

13.2 Approximation algorithms

One use of a randomized construction is to approximate the solution to an otherwise difficult problem. In this section, we start with a trivial approximation algorithm for the largest cut in a graph, and then show a more powerful randomized approximation algorithm, due to Goemans and Williamson [GW94], that gets a better approximation ratio for a much larger class of problems.

13.2.1 MAX CUT

We've previously seen (§2.3.3.2) a randomized algorithm for finding small cuts in a graph. What if we want to find a large cut?

Here is a particularly brainless algorithm that finds a large cut. For each vertex, flip a coin: if the coin comes up heads, put the vertex in S, otherwise, put in it T. For each edge, there is a probability of exactly $1/2$ that it is included in the S–T cut. It follows that the expected size of the cut is exactly $m/2$.

One consequence of this is that every graph has a cut that includes at least half the edges. Another is that this algorithm finds such a cut, with probability at least $\frac{1}{m+1}$. To prove this, let X be the random variable representing the number of edges in the cut, and let p be the probability that $X \geq m/2$. Then

$$\begin{aligned} m/2 &= \mathrm{E}[X] \\ &= (1-p)\,\mathrm{E}[X \mid X < m/2] + p\,\mathrm{E}[X \mid X \geq m/2] \\ &\leq (1-p)\frac{m-1}{2} + pm. \end{aligned}$$

Solving this for p gives the claimed bound.[4]

By running this enough times to get a good cut, we get a polynomial-time randomized algorithm for approximating the **maximum cut** within a factor of 2, which is pretty good considering that MAX CUT is **NP**-hard.

There exist better approximation algorithms. Goemans and Williamson [GW95] give a 0.87856-approximation algorithm for MAX CUT based on randomized rounding of a semidefinite program.[5] The analysis of this algorithm is a little involved, so we won't attempt to show this here, but we will describe (in §13.2.2) an earlier result, also due to Goemans and Williamson [GW94], that gives a $\frac{3}{4}$-approximation to MAX SAT using a similar technique.

[4] This is tight for $m = 1$, but I suspect it's an underestimate for larger m. The main source of slop in the analysis seems to be the step $\mathrm{E}[X \mid X \geq m/2] \leq m$; using a concentration bound, we should be able to show a much stronger inequality here and thus a much larger lower bound on p.

[5] Semidefinite programs are like linear programs except that the variables are vectors instead of scalars, and the objective function and constraints apply to linear combinations of dot-products of these variables. The Goemans-Williamson MAX CUT algorithm is based on a relaxation of the integer optimization problem of maximizing $\sum x_i x_j$ where each $x_i \in \{-1, +1\}$ encodes membership in S or T. They instead allow x_i to be any unit vector in an n-dimensional space, and then take the sign of the dot-product with a random unit vector r to map each optimized x_i to one side of the cut or the other.

13.2.2 MAX SAT

Like MAX CUT, MAX SAT is an **NP**-hard optimization problem that sounds like a very short story about Max. We are given a **satisfiability problem** in **conjunctive normal form**: as a conjunction (AND) of m **clauses**, each of which is the OR of a bunch of variables or their negations. We want to choose values for the n variables that **satisfy** as many of the clauses as possible, where a clause is satisfied if it contains a true variable or the negation of a false variable.[6]

We can instantly satisfy at least $m/2$ clauses on average by assigning values to the variables independently and uniformly at random; the analysis is the same as in §13.2.1 for large cuts, since a random assignment makes the first variable in a clause true with probability $1/2$. Since this approach doesn't require thinking and doesn't use the fact that many of our clauses may have more than one variable, we can probably do better. Except we can't do better in the worst case, because it might be that our clauses consist entirely of x and $\neg x$ for some variable x; clearly, we can only satisfy half of these. We could do better if we knew all of our clauses consisted of at least k distinct literals (satisfied with probability $1 - 2^{-k}$), but we can't count on this. We also can't count on clauses not being duplicated, so it may turn out that skipping a few hard-to-satisfy small clauses hurts us if they occur many times.

Our goal will be to get a good **approximation ratio**, defined as the ratio between the number of clauses we manage to satisfy and the actual maximum that can be satisfied. The tricky part in designing **approximation algorithms** is showing that the denominator here won't be too big. We can do this using a standard trick, of expressing our original problem as an **integer program** and then **relaxing** it to a **linear program**[7] whose

[6] The presentation here follows [MR95, §5.2], which in turn is mostly based on a classic paper of Goemans and Williamson [GW94].

[7] A **linear program** is an optimization problem where we want to maximize (or minimize) some linear **objective function** of the variables subject to linear-inequality constraints. A simple example would be to maximize $x + y$ subject to $2x + y \leq 1$ and $x + 3y \leq 1$; here the **optimal solution** is the assignment $x = \frac{2}{5}, y = \frac{1}{5}$, which sets the objective function to its maximum possible value $\frac{3}{5}$. An **integer program** is a linear program where some of the variables are restricted to be integers. Determining if an integer program even has a solution is **NP**-complete; in contrast, linear programs can be solved in polynomial time. We can **relax** an integer program to a linear program by dropping the requirements that variables be integers; this will let us find a **fractional solution** that is at least as good as the best integer solution, but might be undesirable because it tells us to do something that is ludicrous in the context of our original problem, like only putting half a passenger on the next plane.

solution doesn't have to consist of integers. We then convert the fractional solution back to an integer solution by rounding some of the variables to integer values randomly in a way that preserves their expectations, a technique known as **randomized rounding**.[8]

Here is the integer program (taken from [MR95, §5.2]). We let $z_j \in \{0, 1\}$ represent whether clause C_j is satisfied, and let $y_i \in \{0, 1\}$ be the value of variable x_i. We also let C_j^+ and C_j^- be the set of variables that appear in C_j with and without negation. The problem is to maximize

$$\sum_{j=1}^{m} z_j$$

subject to

$$\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j$$

for all j.

The main trick here is to encode OR in the constraints; there is no requirement that z_j is the OR of the y_i and $(1 - y_i)$ values, but we maximize the objective function by setting it that way.

Sadly, solving integer programs like the above is **NP**-hard (which is not surprising, since if we could solve this particular one, we could solve SAT). But if we drop the requirements that $y_i, z_j \in \{0, 1\}$ and replace them with $0 \leq y_i \leq 1$ and $0 \leq z_j \leq 1$, we get a linear program—solvable in polynomial time—with an optimal value at least as good as the value for the integer program, for the simple reason that any solution to the integer program is also a solution to the linear program.

The problem now is that the solution to the linear program is likely to be **fractional**: instead of getting useful 0–1 values, we might find out we are supposed to make x_i only 2/3 true. So we need one more trick to turn the

Linear programming has an interesting history. The basic ideas were developed independently by Leonid Kantorovich in the Soviet Union and George Dantzig in the United States around the start of the Second World War. Kantorovich's work had direct relevance to Soviet planning problems, but wasn't pursued seriously because it threatened the political status of the planners, required computational resources that weren't available at the time, and looked suspiciously like trying to sneak a capitalist-style price system into the planning process; for a fictionalized account of this tragedy, see [Spu12]. Dantzig's work, which included the development of the **simplex method** for solving linear programs, had a higher impact, although its publication was delayed until 1947 by wartime secrecy.

[8] Randomized rounding was invented by Raghavan and Thompson [RT87]; the particular application here is due to Goemans and Williamson [GW94].

fractional values back into integers. This is the randomized rounding step: given a fractional assignment $\hat{y}_i$, we set x_i to true with probability $\hat{y}_i$.

So what does randomized rounding do to clauses? In our fractional solution, a clause might have value $\hat{z}_j$, obtained by summing up bits and pieces of partially-true variables. We'd like to argue that the rounded version gives a similar probability that C_j is satisfied.

Suppose C_j has k variables; to make things simpler, we'll pretend that C_j is exactly $x_1 \vee x_2 \vee \ldots x_k$. Then the probability that C_j is satisfied is exactly $1 - \prod_{i=1}^{k}(1 - \hat{y}_i)$. This quantity is minimized subject to $\sum_{i=1}^{k} \hat{y}_i \geq \hat{z}_j$ by setting all $\hat{y}_i$ equal to $\hat{z}_j/k$ (easy application of Lagrange multipliers, or can be shown using a convexity argument). Writing z for $\hat{z}_j$, this gives

$$\begin{aligned}
\Pr\left[C_j \text{ is satisfied}\right] &= 1 - \prod_{i=1}^{k}(1 - \hat{y}_i) \\
&\geq 1 - \prod_{i=1}^{k}(1 - z/k) \\
&= 1 - (1 - z/k)^k \\
&\geq z(1 - (1 - 1/k)^k). \\
&\geq z(1 - 1/e).
\end{aligned}$$

The second-to-last step looks like a typo, but it actually works. The idea is to observe that the function $f(z) = 1 - (1 - z/k)^k$ is concave (Proof: $\frac{d^2}{dz^2} f(z) = -\frac{k-1}{k}(1 - z/k)^{k-2} < 0$), while $g(z) = z(1 - (1 - 1/k)^k)$ is linear, so since $f(0) = 0 = g(0)$ and $f(1) = 1 - (1 - 1/k)^k = g(1)$, any point in between must have $f(z) \geq g(z)$. An example of this argument for $k = 3$ is depicted in Figure 13.1.

Since each clause is satisfied with probability at least $\hat{z}_j(1 - 1/e)$, the expected number of satisfied clauses is at least $(1 - 1/e)\sum_j \hat{z}_j$, which is at least $(1 - 1/e)$ times the optimum. This gives an approximation ratio of slightly more than 0.632, which is better than 1/2, but still kind of weak.

So now we apply the second trick from [GW94]: we'll observe that, on a per-clause basis, we have a randomized rounding algorithm that is good at satisfying small clauses (the coefficient $(1 - (1 - 1/k)^k)$ goes all the way up to 1 when $k = 1$), and our earlier dumb algorithm that is good at satisfying big clauses. We can't combine these directly (the two algorithms demand different assignments), but we can run both in parallel and take whichever gives a better result.

To show that this works, let X_j be indicator for the event that clause j is satisfied by the randomized-rounding algorithm and Y_j the indicator for the

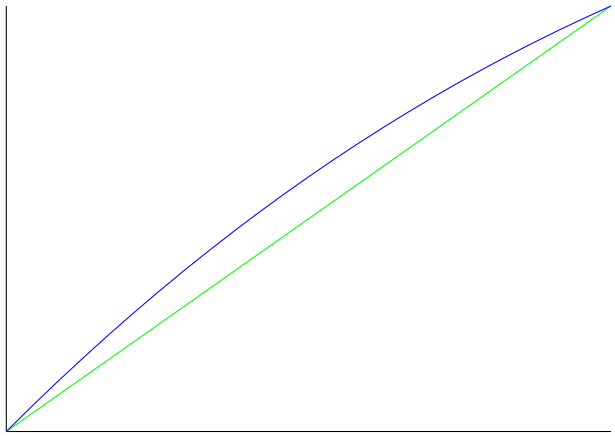


Figure 13.1: Tricky step in MAX SAT argument, showing $(1-(1-z/k)^k) \geq z(1-(1-1/k)^k)$ when $0 \leq z \leq 1$. The case $k = 3$ is depicted.

even that it is satisfied by the simple algorithm. Then if C_j has k literals,

$$\begin{aligned}\mathrm{E}[X_j] + \mathrm{E}[Y_j] &\geq (1-2^{-k}) + (1-(1-1/k)^k)\hat{z}_j \\ &\geq \Big((1-2^{-k}) + (1-(1-1/k)^k)\Big)\hat{z}_j \\ &= \Big(2 - 2^{-k} - (1-1/k)^k\Big)\hat{z}_j.\end{aligned}$$

The coefficient here is exactly $3/2$ when $k = 1$ or $k = 2$, and rises thereafter, so for integer k we have $\mathrm{E}[X_j] + \mathrm{E}[Y_j] \geq (3/2)\hat{z}_j$. Summing over all j then gives $\mathrm{E}\left[\sum_j X_j\right] + \mathrm{E}\left[\sum_j Y_j\right] \geq (3/2)\sum_j \hat{z}_j$. But then one of the two expected sums must beat $(3/4)\sum_j \hat{z}_j$, giving us a $(3/4)$-approximation algorithm.

13.3 The Lovász Local Lemma

Suppose we have a finite set of bad events $\mathcal{A}$, and we want to show that with nonzero probability, none of these events occur. Formally, we want to show $\Pr\left[\bigcap_{A\in\mathcal{A}} \bar{A}\right] > 0$.

Our usual trick so far has been to use the union bound (4.2.1) to show that $\sum_{A\in\mathcal{A}} \Pr[A] < 1$. But this only works if the events are actually improbable. If the union bound doesn't work, we might be lucky enough to have the events be independent; in this case, $\Pr\left[\bigcap_{A\in\mathcal{A}} \bar{A}\right] = \prod_{A\in\mathcal{A}} \Pr\left[\bar{A}\right] > 0$, as long as each event $\bar{A}$ occurs with positive probability. But most of the time,

we'll find that the events we care about aren't independent, so this won't work either.

The **Lovász Local Lemma** [EL75] handles a situation intermediate between these two extremes, where events are generally not independent of each other, but each collection of events that are not independent of some particular event A has low total probability. In the original version, it's non-constructive: the lemma shows a nonzero probability that none of the events occur, but this probability may be very small if we try to sample the events at random, and there is no guidance for how to find a particular outcome that makes all the events false.

Subsequent work [Bec91, Alo91, MR98, CS00, Sri08, Mos09, MT10] showed how, when the events A are determined by some underlying set of independent variables and independence between two events is detected by having non-overlapping sets of underlying variables, an actual solution could be found in polynomial expected time. The final result in this series, due to Moser and Tardos [MT10], gives the same bounds as in the original non-constructive lemma, using the simplest algorithm imaginable: whenever some bad event A occurs, try to get rid of it by resampling all of its variables, and continue until no bad events are left.

13.3.1 General version

A formal statement of the general lemma is:[9]

Lemma 13.3.1. *Let $\mathcal{A} = A_1, \ldots, A_m$ be a finite collection of events on some probability space, and for each $A \in \mathcal{A}$, let $\Gamma(A)$ be a set of events such that A is independent of all events not in $\Gamma^+(A) = \{A\} \cup \Gamma(A)$. If there exist real numbers $0 < x_A < 1$ such that, for all events $A \in \mathcal{A}$*

$$\Pr[A] \leq x_A \prod_{B \in \Gamma(A)} (1 - x_B), \tag{13.3.1}$$

then

$$\Pr\left[\bigcap_{A \in \mathcal{A}} \bar{A}\right] \geq \prod_{A \in \mathcal{A}} (1 - x_A). \tag{13.3.2}$$

In particular, this means that the probability that none of the A_i occur is not zero, since we have assumed $x_{A_i} < 1$ holds for all i.

The role of x_A in the original proof is to act as an upper bound on the probability that A occurs given that some collection of other events doesn't

[9] This version is adapted from [MT10].

occur. For the constructive proof, the x_A are used to show a bound on the number of resampling steps needed until none of the A occur.

13.3.2 Symmetric version

For many applications, having to come up with the x_A values can be awkward. The following symmetric version is often used instead:

Corollary 13.3.2. *Let $\mathcal{A}$ and Γ be as in Lemma 13.3.1. Suppose that there are constants p and d, such that for all $A \in \mathcal{A}$, we have $\Pr[A] \leq p$ and $|\Gamma(A)| \leq d$. Then if $ep(d+1) < 1$, $\Pr\left[\bigcap_{A\in\mathcal{A}} \bar{A}\right] \neq 0$.*

Proof. Basically, we are going to pick a single value x such that $x_A = x$ for all A in $\mathcal{A}$, and (13.3.1) is satisfied. This works as long as $p \leq x(1-x)^d$, as in this case we have, for all A, $\Pr[A] \leq p \leq x(1-x)^d \leq x(1-x)^{|\Gamma(A)|} = x_A\left(\prod_{B\in\Gamma(A)}(1-x_B)\right)$.

For fixed d, $x(1-x)^d$ is maximized using the usual trick: $\frac{d}{dx}x(1-x)^d = (1-x)^d - xd(1-x)^{d-1} = 0$ gives $(1-x) - xd = 0$ or $x = \frac{1}{d+1}$. So now we need $p \leq \frac{1}{d+1}\left(1-\frac{1}{d+1}\right)^d$. It is possible to show that $1/e < \left(1-\frac{1}{d+1}\right)^d$ for all $d \geq 0$.[10] So $ep(d+1) \leq 1$ implies $p \leq \frac{1}{e(d+1)} \leq \left(1-\frac{1}{d+1}\right)^d \frac{1}{d+1} \leq x(1-x)^{|\Gamma(A)|}$ as required by Lemma 13.3.1. □

13.3.3 Applications

Here we give some simple applications of the lemma.

13.3.3.1 Graph coloring

Let's start with a problem where we know what the right answer should be. Suppose we want to color the vertices of a cycle with c colors, so that no edge has two endpoints with the same color. How many colors do we need?

Using brains, we can quickly figure out that $c = 3$ is enough. Without brains, we could try coloring the vertices randomly: but in a cycle with n vertices and n edges, on average n/c of the edges will be monochromatic, since each edge is monochromatic with probability $1/c$. If these bad events were independent, we could argue that there was a $(1-1/c)^n > 0$ probability

[10] Observe that $\lim_{d\to\infty}\left(1-\frac{1}{d+1}\right)^d = e^{-1}$ and that $f(d) = \left(1-\frac{1}{d+1}\right)^d = e^{-1}$ is a decreasing function. The last part can be shown by taking the derivative of $\log f(d)$. See the solution to Problem G.1.4 for the gory details.

that none of them occurred, but they aren't, so we can't. Instead, we'll use the local lemma.

The set of bad events $\mathcal{A}$ is just the set of events $A_i = [\text{edge } i \text{ is monochromatic}]$. We've already computed $p = 1/c$. To get d, notice that each edge only shares a vertex with two other edges, so $|\Gamma(A_i)| \leq 2$. Corollary 13.3.2 then says that there is a good coloring as long as $ep(d+1) = 3e/c \leq 1$, which holds as long as $c \geq 9$. We've just shown we can 9-color a cycle. If use the asymmetric version, we can set all x_A to $1/3$ and show that $p \leq \frac{1}{3}\left(1 - \frac{1}{3}\right)^2 = \frac{4}{27}$ would also work; with this we can 7-color a cycle. This is still not as good as what we can do if we are paying attention, but not bad for a procedure that doesn't use the structure of the problem much.

13.3.3.2 Satisfiability of k-CNF formulas

A more sophisticated application is demonstrating satisfiability for k-CNF formulas where each variable appears in a bounded number of clauses. Recall that a **k-CNF formula** consists of m **clauses**, each of which consists of exactly k variables or their negations (collectively called **literals**. It is **satisfied** if at least one literal in every clause is assigned the value true.

Suppose that each variable appears in at most ℓ clauses. Let $\mathcal{A}$ consist of all the events $A_i = [\text{clause } i \text{ is not satisfied}]$. Then, for all i, $\Pr[A_i] = 2^{-k}$ exactly, and $|\Gamma(A_i)| \leq d = k(\ell - 1)$ since each variable in A_i is shared with at most $\ell - 1$ other clauses, and A_i will be independent of all events A_j with which it doesn't share a variable. So if $ep(d+1) = e2^{-k}(k(\ell-1)+1) < 1$, which holds if $\ell < \frac{2^k + e(k-1)}{ek}$, Corollary 13.3.2 tells us that a satisfying assignment exists.[11]

Corollary 13.3.2 doesn't let us actually find a satisfying assignment, but it turns out we can do that too. We'll return to this when we talk about the constructive version of the local lemma in §13.3.5.

13.3.3.3 Hypergraph 2-colorability

This was the original motivation for the lemma. A **k-uniform hypergraph** $G = \langle V, E\rangle$ has a set of **hyperedges** E, each of which contains exactly k vertices from V. A **2-coloring** of the hypergraph assigns one of two colors to each vertex in v, so that no edge is monochromatic.

Suppose we color vertices at random. Pick some edge S, and let A_S be the probability that it is monochromatic. Then $\Pr[A_S] = 2^{1-k}$ exactly. We

[11] To avoid over-selling this claim, it's worth noting that the bound on ℓ only reaches 2 at $k = 4$, although it starts growing pretty fast after that.

also have that A_S depends only on events A_T where $S \cap T \neq \emptyset$. Suppose that there is a bound d on the number of hyperedges T that overlap with any hyperedge S. Then Corollary 13.3.2 says that there exists a good coloring as long as $e2^{1-k}(d+1) < 1$. This holds for any hypergraph with $d < 2^{k-1}/e - 1$.

Unlike graph coloring, it's not so obvious how to solve hypergraph coloring with a simple greedy algorithm. So here the local lemma seems to buy us something we didn't already have.

13.3.4 Non-constructive proof

This is essentially the same argument presented in [MR95, §5.5], but we adapt the notation to match the statement in terms of neighborhoods $\Gamma(A)$ instead of edges in a **dependency graph**, where an event is independent of all events that are not its successors in the graph. The two formulations are identical, since we can always represent the neighborhoods $\Gamma(A)$ by creating an edge from A to B when B is in $\Gamma(A)$; and conversely we can convert a dependency graph into a neighborhood structure by making $\Gamma(A)$ the set of successors of A in the dependency graph.

The proof is a bit easier than the constructive version, while also being a bit more general because the neighborhood structure doesn't depend on identifying a collection of underlying independent variables.

The idea is to order the events in $\mathcal{A}$ as $A_1, A_2, \ldots, A_n$ and expand $\Pr\left[\bigcap_{i=1}^n \bar{A}_i\right]$ as $\prod_{i=1}^n \Pr\left[\bar{A}_i \,\middle|\, \bigcap_{j=1}^{i-1} \bar{A}_j\right]$. If we can show that every factor in the product is nonzero, we are done.

To do this, we will show by induction on $|S|$ the more general statement that for any A and *any* $S \subseteq \mathcal{A}$ with $A \notin S$,

$$\Pr\left[A \,\middle|\, \bigcap_{B \in S} \bar{B}\right] \leq x_A. \tag{13.3.3}$$

When $|S| = 0$, this just says $\Pr[A] \leq x_A$, which follows immediately from (13.3.1).

For larger S, split S into $S_1 = S \cap \Gamma(A)$, the events in S that might not be independent of A; and $S_2 = S \setminus \Gamma(A)$, the events in S that we know to be independent of A. If $S_2 = S$, then A is independent of all events in S, and (13.3.3) follows immediately from $\Pr\left[A \,\middle|\, \bigcap_{B\in S} \bar{B}\right] = \Pr[A] \leq x_A \prod_{B\in\Gamma(A)}(1 - x_B) \leq x_A$. Otherwise $|S_2| < |S|$, which means that we can assume the induction hypothesis holds for S_2.

Write C_1 for the event $\bigcap_{B\in S_1} \bar{B}$ and C_2 for the event $\bigcap_{B\in S_2} \bar{B}$. Then

$\bigcap_{B\in S}\bar{B} = C_1 \cap C_2$ and we can expand

$$\begin{aligned}\Pr\left[A \,\middle|\, \bigcap_{B\in S}\bar{B}\right] &= \frac{\Pr[A\cap C_1\cap C_2]}{\Pr[C_1\cap C_2]}\\ &= \frac{\Pr[A\cap C_1 \mid C_2]\Pr[C_2]}{\Pr[C_1\mid C_2]\Pr[C_2]}\\ &= \frac{\Pr[A\cap C_1\mid C_2]}{\Pr[C_1\mid C_2]}. \end{aligned} \tag{13.3.4}$$

We don't need to do anything particularly clever with the numerator:

$$\begin{aligned}\Pr[A\cap C_1\mid C_2] &\le \Pr[A\mid C_2]\\ &= \Pr[A]\\ &\le x_A\prod_{B\in\Gamma(A)}(1-x_B), \end{aligned} \tag{13.3.5}$$

from (13.3.1) and the fact that A is independent of all B in S_2 and thus also independent of C_2.

For the denominator, we expand C_1 back out to $\bigcap_{B\in S_1}\bar{B}$ and break out the induction hypothesis. To bound $\Pr\left[\bigcap_{B\in S_1}\bar{B}\,\middle|\, C_2\right]$, we order S_1 arbitrarily as $\{B_1,\ldots,B_r\}$, for some r, and show by induction on ℓ as ℓ goes from 1 to r that

$$\Pr\left[\bigcap_{i=1}^{\ell}\bar{B}_i \,\middle|\, C_2\right] \ge \prod_{i=1}^{\ell}(1-x_{B_i}). \tag{13.3.6}$$

The proof is that, for $\ell = 1$,

$$\begin{aligned}\Pr\left[\bar{B}_1 \,\middle|\, C_2\right] &= 1-\Pr[B_1\mid C_2]\\ &\ge 1-x_{B_1}\end{aligned}$$

using the outer induction hypothesis (13.3.3), and for larger ℓ, we can compute

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{\ell}\bar{B}_i \,\middle|\, C_2\right] &= \Pr\left[\bar{B}_\ell \,\middle|\, \left(\bigcap_{i=1}^{\ell-1}\bar{B}_i\right)\cap C_2\right]\cdot\Pr\left[\bigcap_{i=1}^{\ell-1}\bar{B}_i \,\middle|\, C_2\right]\\ &\ge (1-x_{B_\ell})\prod_{i=1}^{\ell-1}(1-x_{B_i})\\ &= \prod_{i=1}^{\ell}(1-x_{B_i}),\end{aligned}$$

where the second-to-last step uses the outer induction hypothesis (13.3.3) for the first term and the inner induction hypothesis (13.3.6) for the rest. This completes the proof of the inner induction.

When $\ell = r$, we get

$$\begin{aligned}\Pr\left[C_1 \mid C_2\right] &= \Pr\left[\bigcap_{i=1}^{r} \bar{B}_i \,\middle|\, C_2\right] \\ &\geq \prod_{B \in S_1} (1 - x_B). \end{aligned} \tag{13.3.7}$$

Substituting (13.3.5) and (13.3.7) into (13.3.4) gives

$$\begin{aligned}\Pr\left[A \,\middle|\, \bigcap_{B \in S} \bar{B}\right] &\leq \frac{x_A \left(\prod_{B \in \Gamma(A)} (1 - x_B)\right),}{\prod_{B \in S_1} (1 - x_B)} \\ &= x_A \left(\prod_{B \in \Gamma(A) \setminus S_1} (1 - x_B)\right) \\ &\leq x_A.\end{aligned}$$

This completes the proof of the outer induction.

To get the bound (13.3.2), we reach back inside the proof and repeat the argument for (13.3.7) with $\bigcap_{A \in \mathcal{A}} \bar{A}$ in place of C_1 and without the conditioning on C_2. We order $\mathcal{A}$ arbitrarily as $\{A_1, A_2, \ldots, A_m\}$ and show by induction on k that

$$\Pr\left[\bigcap_{i=1}^{k} \bar{A}_i\right] \geq \prod_{i=1}^{k} (1 - x_{A_k}). \tag{13.3.8}$$

For the base case we have $k = 0$ and $\Pr[\Omega] \geq 1$, using the usual conventions on empty products. For larger k, we have

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{k} \bar{A}_i\right] &= \Pr\left[\bar{A}_k \,\middle|\, \bigcap_{i=1}^{k-1} \bar{A}_i\right] \Pr\left[\bigcap_{i=1}^{k-1} \bar{A}_i\right] \\ &\geq (1 - x_{A_k}) \prod_{i=1}^{k-1} (1 - x_{A_i}) \\ &\geq \prod_{i=1}^{k} (1 - x_{A_k}),\end{aligned}$$

where in the second-to-last step we use (13.3.3) for the first term and the induction hypothesis (13.3.8) for the big product.

Setting $k = m$ finishes the proof.

13.3.5 Constructive proof

We now describe he constructive proof of the Lovász local lemma due to Moser and Tardos [MT10], which is based on a slightly more specialized construction of Moser alone [Mos09]. This version applies when our set of bad events $\mathcal{A}$ is defined in terms of a set of *independent* variables $\mathcal{P}$, where each $A \in \mathcal{A}$ is determined by some set of variables $\text{vbl}(A) \subseteq \mathcal{P}$, and $\Gamma(A)$ is defined to be the set of all events $B \neq A$ that share at least one variable with A; i.e., $\Gamma(A) = \{B \in \mathcal{A} \setminus \{A\} \mid \text{vbl}(B) \cap \text{vbl}(A) \neq \emptyset\}$.

In this case, we can attempt to find an assignment to the variables that makes none of the A occur using the obvious algorithm of sampling an initial state randomly, then resampling all variables in $\text{vbl}(A)$ whenever we see some bad A occur. Astonishingly, this actually works in a reasonable amount of time, without even any cleverness in deciding which A to resample at each step, if the conditions for Lemma 13.3.1 hold for x that are not too large. In particular, we will show that a good assignment is found after each A is resampled at most $\frac{x_A}{1-x_A}$ times on average.[12]

Lemma 13.3.3. *Under the conditions of Lemma 13.3.1, the Moser-Tardos procedure does at most*

$$\sum_A \frac{x_A}{1 - x_A}$$

resampling steps on average.

We defer the proof of Lemma 13.3.3 for the moment. For most applications, the following symmetric version is easier to work with:

Corollary 13.3.4. *Under the conditions of Corollary 13.3.2, the Moser-Tardos procedure does at most m/d resampling steps on average.*

Proof. Follows from Lemma 13.3.3 and the choice of $x_A = \frac{1}{d+1}$ in the proof of Corollary 13.3.2. □

How this expected m/d bound translates into actual time depends on the cost of each resampling step. The expensive part at each step is likely to be the cost of finding an A that occurs and thus needs to be resampled.

[12] Even though there is a lot of resampling going on here, Moser-Rardos is not actually a sampling algorithm, in the sense that some solutions to the original problem may be much more likely to be produced than others. There has been some more recent work on the **sampling Lovász local lemma**, where the goal is to obtain a close-to-uniform sample from the space of possible solutions. This is much harder than just finding one solution and usually requires stronger constraints on the problem. An example of this work is the paper of He *et al.* [HWY22].

Intuitively, we might expect the resampling to work because if each $A \in \mathcal{A}$ has a small enough neighborhood $\Gamma(A)$ and a low enough probability, then whenever we resample A's variables, it's likely that we fix A and unlikely that we break too many B events in A's neighborhood. It turns out to be tricky to quantify how this process propagates outward, so the actual proof uses a different idea that essentially looks at this process in reverse, looking for each resampled event A at a set of previous events whose resampling we can blame for making A occur, and then showing that this tree (which will include every resampling operation as one of its vertices) can't be too big.

The first step is to fix some strategy for choosing which event A to resample at each step. We don't care what this strategy is; we just want it to be the case that the sequence of events depends only on the random choices made by the algorithm in its initial sampling and each resampling. We can then define an **execution log** C that lists the sequence of events $C_1, C_2, C_3, \ldots$ that are resampled at each step of the execution.

From C we construct a **witness tree** T_t for each resampling step t whose nodes are labeled with events, with the property that the children of a node v labeled with event A_v are labeled with events in $\Gamma^+(A_v) = \{A_v\} \cup \Gamma(A_v)$.

The root of T_t is labeled with C_t. To construct the rest of the tree, we work backwards through $C_{t-1}, C_{t-2}, \ldots, C_1$, and for each event C_i we encounter, we attach C_i as a child of the deepest v we can find with $C_i \in \Gamma^+(A_v)$, choosing arbitrarily if there is more than one such v, and discarding C_i if there is no such v.

Now we can ask what the probability is that we see some particular witness tree T in the execution log. Each vertex of T corresponds to some event A_v that we resample because it occurs; in order for it to occur, the previous assignments of each variable in $\text{vbl}(A_v)$ must have made A_v true, which occurs with probability $\Pr[A_v]$. But since we resample all the variables in A_v, any subsequent assignments to these variables are independent of the ones that contributed to v; with sufficient handwaving (or a rather detailed coupling argument as found in [MT10]) this gives that each event A_v occurs with independent probability $\Pr[A_v]$, giving $\Pr[T] = \prod_{v \in T} \Pr[A_v]$.

Why do we care about this? Because every event we resample is the root of some witness tree, and we can argue that every event we resample is the root of a *distinct* witness tree. The proof is that since we only discard events B that have $\text{vbl}(B)$ disjoint from all nodes already in the tree, once we put A at the root, any other instance of A gets included. So the witness tree rooted at the i-th occurrence of A in C will include exactly i copies of A, unlike the witness tree rooted at the j-th copy for $j \neq i$.

Now comes the sneaky trick: we'll bound how many distinct witness

trees T we expect to see rooted at A, given that each occurs with probability $\prod_{v \in T} \Pr[A_v]$. This is done by constructing a **branching process** using the x_B values from Lemma 13.3.1 as probabilities of a node with label A having a child labeled B for each B in $\Gamma^+(A)$, and doing algebraic manipulations on the resulting probabilities until $\prod_{v \in T} \Pr[A_v]$ shows up. We can then sum over the expected number of copies of trees to get a bound on the expected number of events in the execution log (since each such even is the root of some tree), which is equal to the expected number of resamplings.

Consider the process where we start with a root labeled A, and for each vertex v with label A_v, give it a child labeled B for each $B \in \Gamma^+(A_v)$ with independent probability x_B. We'll now calculate the probability p_T that this process generates a particular tree T in the set $\mathcal{T}_A$ of trees with root A.

Let $x'_B = x_B \prod_{C \in \Gamma(B)} (1 - x_C)$. Note that (13.3.1) says precisely that $\Pr[B] \leq x'_B$.

For each vertex v in T, let $W_v \subseteq \Gamma^+(A_v)$ be the set of events $B \in \Gamma^+(A_v)$ that *don't* occur as labels of children of v. The probability of getting T is equal to the product of the probabilities at each v of getting all of its children and none of its non-children. The non-children of v collectively contribute $\prod_{B \in W_v} (1 - x_B)$ to the product, and v itself contributes x_{A_v} (via the product for its parent), unless v is the root node. So we can express the giant product as

$$p_T = \frac{1}{x_A} \prod_{v \in T} \left(x_{A_v} \prod_{B \in W_v} (1 - x_B) \right).$$

We don't like the W_v very much, so we get rid of them by taking the product of B in $\Gamma^+(A)$, then dividing out the ones that aren't in W_v. This gives

$$\begin{aligned} p_T &= \frac{1}{x_A} \prod_{v \in T} \left(x_{A_v} \prod_{B \in W_v} (1 - x_B) \right). \\ &= \frac{1}{x_A} \prod_{v \in T} \left(x_{A_v} \left(\prod_{B \in \Gamma^+(A_v)} (1 - x_B) \right) \left(\prod_{B \in \Gamma^+(A_v) \setminus W_v} \frac{1}{1 - x_B} \right) \right). \end{aligned}$$

This seems like we exchanged one annoying index set for another, but each element of $\Gamma^+(A_V) \setminus W_v$ is $A_{v'}$ for some child of v in T. So we can push these factors down to the children, and since we are multiplying over all vertices in T, they will each show up exactly once except at the root. To keep the products clean, we'll throw in $\frac{1}{1 - x_A}$ for the root as well, but compensate for

this by multiplying by $1 - x_A$ on the outside of the product. This gives

$$\begin{aligned} p_T &= \frac{1 - x_A}{x_A} \prod_{v \in T} \left(\frac{x_{A_v}}{1 - x_{A_v}} \prod_{B \in \Gamma^+(A_v)} (1 - x_B) \right) \\ &= \frac{1 - x_A}{x_A} \prod_{v \in T} \left(x_{A_v} \prod_{B \in \Gamma(A_v)} (1 - x_B) \right) \\ &= \frac{1 - x_A}{x_A} \prod_{v \in T} x'_{A_v}. \end{aligned}$$

Now we can bound the expected number of trees rooted at A that appear in C, assuming (13.3.1) holds. Letting $\mathcal{T}_A$ as before be the set of all such trees and N_A the number that appear in C, we have

$$\begin{aligned} \mathrm{E}[N_A] &= \sum_{T \in \mathcal{T}_A} \Pr[T \text{ appears in } C] \\ &\leq \sum_{T \in \mathcal{T}_A} \prod_{v \in T} \Pr[A(v)] \\ &\leq \sum_{T \in \mathcal{T}_A} \prod_{v \in T} x'_{A_v} \\ &= \sum_{T \in \mathcal{T}_A} \frac{x_A}{1 - x_A} p_T \\ &= \frac{x_A}{1 - x_A} \sum_{T \in \mathcal{T}_A} p_T \\ &\leq \frac{x_A}{1 - x_A}. \end{aligned}$$

The last sum is bounded by one because occurrences of particular trees T in $\mathcal{T}_A$ are all disjoint events.

Now sum over all A, and we're done.

Chapter 14

Derandomization

Derandomization is the process of taking a randomized algorithm and turning it into a deterministic algorithm. This is useful both for practical reasons (deterministic algorithms are more predictable, which makes them easier to debug and gives hard guarantees on running time) and theoretical reasons (if we can derandomize any randomized algorithm we could show results like $\mathbf{P} = \mathbf{RP}$,[1] which would reduce the number of complexity classes that complexity theorists otherwise have to deal with). It may also be the case that derandomizing a randomized algorithm can be used for **probability amplification**, where we replace a low probability of success with a higher probability, in this case 1.

There are basically two approaches to derandomization:

1. Reduce the number of random bits used down to $O(\log n)$, and then search through all choices of random bits exhaustively. For example, if we only need pairwise independence, we could use the XOR technique from §5.1.2.1 to replace a large collection of variables with a small collection of random bits.

   Except for the exhaustive search part, this is how randomized algorithms are implemented in practice: rather than burning random bits continuously, a **pseudorandom generator** is initialized from a **seed** consisting of a small number of random bits. For pretty much all of the randomized algorithms we know about, we don't even need to use a particularly strong pseudorandom generator. This is largely because

[1] The class **RP** consists of all languages L for which there is a polynomial-time randomized algorithm that correctly outputs "yes" given an input x in L with probability at least $1/2$, and never answers "yes" given an input x not in L. See §1.5.2 for a more extended description of **RP** and other randomized complexity classes.

current popular generators are the products of a process of evolution: pseudorandom generators that cause wonky behavior or fail to pass tests that approximate the assumptions made about them by typical randomized algorithms are abandoned in favor of better generators.[2]

From a theoretical perspective, pseudorandom generators offer the possibility of eliminating randomization from all randomized algorithms, except there is a complication. While (under reasonable cryptographic assumptions) there exist **cryptographically secure pseudorandom generators** whose output is indistinguishable from a genuinely random source by polynomial-time algorithms (including algorithms originally intended for other purposes), such generators are inherently incapable of reducing the number of random bits down to the $O(\log n)$ needed for exhaustive search. The reason is that any pseudorandom generator with only polynomially-many seeds can't be cryptographically secure, because we can distinguish it from a random source by just checking its output against the output for all possible seeds. Whether there is some other method for transforming an arbitrary algorithm in **RP** or **BPP** into a deterministic algorithm remains an open problem in complexity theory (and beyond the scope of this course).

2. Start with a specific randomized protocol and analyze its behavior enough that we can replace the random bits it uses with specific, deterministically-chosen bits we can compute. This is the main approach we will describe below. A non-constructive variant of this shows that we can always replace the random bits used by all inputs of a given size with a few carefully-selected fixed sequences (Adleman's Theorem, described in §14.2). More practical is the **method of conditional probabilities**, which chooses random bits sequentially based on which value is more likely to give a good outcome (see §14.4).

14.1 Deterministic vs. randomized algorithms

In thinking about derandomization, it can be helpful to have more than one way to look at a randomized algorithm. So far, we've describe randomized algorithms as random choices of deterministic algorithms ($M_r(x)$) or,

[2] Having cheaper computers helps here as well. Nobody would have been willing to spend 2496 bytes on the state vector for Mersenne Twister [MN98] back in 1975, but by 1998 this amount of memory was trivial for pretty much any computing device except the tiniest microcontrollers.

equivalently, as deterministic algorithms that happen to have random inputs ($M(r, x)$). This gives a very static view of how randomness is used in the algorithm. A more dynamic view is obtained by thinking of the computation of a randomized algorithm as a **computation tree**, where each path through the tree corresponds to a computation with a fixed set of random bits and a branch in the tree corresponds to a random decision. In either case we want an execution to give us the right answer with reasonably high probability, whether that probability measures our chance of getting a good deterministic machine for our particular input or landing on a good computation path.

14.2 Adleman's theorem

The idea of picking a good deterministic machine is the basis for **Adleman's theorem**[Adl78], a classic result in complexity theory. Adleman's theorem says that we can always replace randomness by an oracle that presents us with a fixed string of **advice** p_n that depends only on the size of the input n and has size polynomial in n. The formal statement of the theorem relates the class **RP**, which is the class of problems for which there exists a polynomial-time Turing machine $M(x, r)$ that outputs 1 at least half the time when $x \in L$ and never when $x \notin L$; and the class **P/poly**, which is the class of problems for which there is a polynomial-sized string p_n for each input size n and a polynomial-time Turing machine M' such that $M'(x, p_{|x|})$ outputs 1 if and only if $x \in L$.

Theorem 14.2.1. **RP** $\subseteq$ **P/poly**.

Proof. The intuition is that if any one random string has a constant probability of making M happy, then by choosing enough random strings we can make the probability that M fails using on every random string for any given input so small that even after we sum over all inputs of a particular size, the probability of failure is still small using the union bound (4.2.1). This is an example of **probability amplification**, where we repeat a randomized algorithm many times to reduce its failure probability.

Formally, consider any fixed input x of size n, and imagine running M repeatedly on this input with $n + 1$ independent sequences of random bits $r_1, r_2, \ldots, r_{n+1}$. If $x \notin L$, then $M(x, r_i)$ never outputs 1. If $x \in L$, then for each r_i, there is an independent probability of at least $1/2$ that $M(x, r_i) = 1$. So $\Pr[M(x, r_i) = 0] \leq 1/2$, and $\Pr[\forall i M(x, r_i) = 0] \leq 2^{-(n+1)}$. If we sum this probability of failure for each individual $x \in L$ of length n over the at most 2^n such elements, we get a probability that any of them fail of at most

$2^n 2^{-(n+1)} = 1/2$. Turning this upside down, any sequence of $n+1$ random inputs includes a **witness** that $x \in L$ for *all* inputs x with probability at least $1/2$. It follows that a good sequence $r_1, \ldots, r_{n+1}$, exists.

Our advice p_n is now some good sequence $p_n = \langle r_1 \ldots r_{n+1} \rangle$, and the deterministic advice-taking algorithm that uses it runs $M(x, r_i)$ for each r_i and returns true if and only if at least one of these executions returns true. □

The classic version of this theorem shows that anything you can do with a polynomial-size randomized circuit (a circuit made up of AND, OR, and NOT gates where some of the inputs are random bits, corresponding to the r input to M) can be done with a polynomial-size deterministic circuit (where now the p_n input is baked into the circuit, since we need a different circuit for each size n anyway).

A limitation of this result is that ordinary algorithms seem to be better described by **uniform** families of circuits, where there exists a polynomial-time algorithm that, given input n, outputs the circuit C_n for processing size-n inputs. In contrast, the class of circuits generated by Adleman's theorem is most likely **non-uniform**: the process of finding the good witnesses r_i is not something we know how to do in polynomial time (with the usual caveat that we can't prove much about what we can't do in polynomial time).

14.3 Limited independence

For some algorithms, it may be that full independence is not needed for all of the random bits. If the amount of independence needed is small enough, this may allow us to reduce the actual number of random bits consumed down to $O(\log n)$, at which point we can try all possible sequences of random bits in polynomial time.

Variants of this technique have been used heavily in the cryptography and complexity; see [LW05] for a survey of some early work in this area. We'll do a quick example of the method before before moving onto more direct approaches.

14.3.1 MAX CUT

Let's look at the randomized MAX CUT algorithm from §13.2.1. In the original algorithm, we use n independent random bits to assign the n vertices to S or T. The idea is that by assigning each vertex independently at

random to one side of the cut or the other, each edge appears in the cut with probability $1/2$, giving a total of $m/2$ edges in the cut in expectation.

Suppose that we replace these n independent random bits with n pairwise-independent bits generated by taking XORs of subsets of $\lceil \lg(n+1) \rceil$ independent random bits as described in §5.1.2.1. Because the bits are pairwise-independent, the probability that the two endpoints of an edge are assigned to different sides of the cut is still exactly $1/2$. So on average we get $m/2$ edges in the cut as before, and there is at least one sequence of random bits that guarantees a cut at least this big.

But with only $\lceil \lg(n+1) \rceil$ random bits, there are only $2^{\lceil \log(n+1) \rceil} < 2(n+1)$ possible sequences of random bits. If we try all of them, then we find a cut of size $m/2$ always. The total cost is $O(n(n+m))$ if we include the $O(n+m)$ cost of testing each cut. Note that this algorithm does not generate all 2^n possible cuts, but among those it *does* generate, there must be a large one.

In this particular case, we'll see below how to get the same result at a much lower cost, using more knowledge of the problem. So we have the typical trade-off between algorithm performance and algorithm designer effort.

14.4 The method of conditional probabilities

The **method of conditional probabilities** [Rag88] follows an execution of the randomized algorithm, but at each point where we would otherwise make a random decision, it makes a decision that minimizes the conditional probability of losing.

Structurally, this is similar to the method of bounded differences (see §5.3.3). Suppose our randomized algorithm generates m random values $X_1, X_2, \ldots, X_m$. Let $f(X_1, \ldots, X_m)$ be the indicator variable for our randomized algorithm failing (more generally, we can make it an expected cost or some other performance measure). Extend f to shorter sequences of values by defining $f(x_1, \ldots, x_k) = \mathrm{E}\left[f(x_1, \ldots, x_k, X_{k+1}, \ldots, X_m)\right]$. Then $Y_k = f(X_1, \ldots, X_k)$ is a Doob martingale, just as in the method of bounded differences. This implies that, for any partial sequence of values $x_1, \ldots, x_k$, there exists some next value x_{k+1} such that $f(x_1, \ldots, x_k) \geq f(x_1, \ldots, x_k, x_{k+1})$. If we can find this value, we can follow a path on which f always decreases, and obtain an outcome of the algorithm $f(x_1, \ldots, x_m)$ less than or equal to the initial value $f(\langle\rangle)$. If our outcomes are 0–1 (as in failure probabilities), and our initial value for f is less than 1, this means that we reach an outcome with $f = 0$.

The tricky part here is that it may be hard to compute $f(x_1, \ldots, x_k)$. (It's always possible to do so in principle by enumerating all assignments of the remaining variables, but if we have time to do this, we can just search for a winning assignment directly.) What makes the method of conditional probabilities practical in many cases is that it's not necessary for f to compute the actual probability of failure, as long as (a) f gives an upper bound on the real probability of failure, at least in terminal configurations, and (b) f has the property used in the argument that for any partial sequence $x_1, \ldots, x_k$ there exists an extension $x_1, \ldots, x_k, x_{k+1}$ with $f(x_1, \ldots, x_k) \geq f(x_1, \ldots, x_k, x_{k+1})$. Such an f is called a **pessimistic estimator**. If we can find a pessimistic estimator that is easy to compute and starts out less than 1, then we can just follow it down the tree to a leaf that doesn't fail.

14.4.1 MAX CUT using conditional probabilities

Again we consider the algorithm from §13.2.1 that assigns vertices to S and T at random. To derandomize this algorithm, at each step we pick a vertex and assign it to the side of the cut that maximizes the conditional expectation of the number of edges that cross the cut. We can compute this conditional expectation as follows:

1. For any edge that already has both endpoints assigned to S or T, it's either in the cut or it isn't: add 0 or 1 as appropriate to the conditional expectation.

2. For any edge with only one endpoint assigned, there's a $1/2$ probability that the other endpoint gets assigned to the other side (in the original randomized algorithm). Add $1/2$ to the conditional expectation for these edges.

3. For any edge with neither endpoint assigned, we again have a $1/2$ probability that it crosses the cut. Add $1/2$ for these as well.

So now let us ask how assigning a particular previously unassigned vertex v to S or T affects the conditional probability. For any neighbor w of v that is not already assigned, adding v to S or T doesn't affect the $1/2$ contribution of vw. So we can ignore these. The only effects we see are that if some neighbor w is in S, assigning v to S decreases the conditional expectation by $1/2$ and assigning v to T increases the expectation by $1/2$. So to maximize the conditional expectation, we should assign v to whichever side currently holds fewer of v's neighbors—the obvious greedy algorithm, which runs in $O(n + m)$ time if we are reasonably clever about keeping track of how many

neighbors each unassigned node has in S and T. The advantage of the method of conditional probabilities here is that we immediately get that the greedy algorithm achieves a cut of size $m/2$, which might require actual intelligence to prove otherwise.

14.4.2 Deterministic construction of Ramsey graphs

Here is an example that is slightly less trivial. For this example we let f be a count of bad events rather than a failure probability, but the same method applies.

Recall from §13.1.2 that if $k \geq 3$, for $n \leq 2^{k/2}$ there exists a graph with n nodes and no cliques or independent sets of size k. The proof of this fact is to observe that each subset of k vertices is bad with probability 2^{-k+1}, and when $\binom{n}{k}2^{-k+1} < 1$, the expected number of bad subsets in $G_{n,1/2}$ is less than 1, showing that some good graph exists.

We can turn this into a deterministic $n^{O(\log n)}$ algorithm for finding a k-Ramsey graph in the worst case when $n = 2^{k/2}$. The trick is to set the edges to be present or absent one at a time, and for each edge, take the value that minimizes the expected number of bad subsets conditioned on the choices so far. We can easily calculate the conditional probability that a subset is bad in $O(k)$ time: if it already has both a present and missing edge, it's not bad. Otherwise, if we've already set ℓ of its edges to the same value, it's bad with probability exactly $2^{-k+\ell}$. Summing this over all $O(n^k)$ subsets takes $O(n^k k)$ time per edge, and we have $O(n^2)$ edges, giving $O(n^{k+2}k) = O\left(n^{(2\lg n+2+\lg\lg n)}\right) = n^{O(\log n)}$ time total.

It's worth mentioning that there are faster deterministic constructions in the literature. The best construction that I am aware of is given by Cohen [Coh21], which constructs a graph with n vertices and no clique or independent set of size $2^{(\log\log n)^c}$ for a particular constant c, in polylogarithmic time per edge.

14.4.3 Derandomized set balancing

Recall the set balancing problem from §13.1.1: given vectors $v_1, v_2, \ldots, v_n$ in $\{0,1\}^m$, we'd like to find ± 1 coefficients $\epsilon_1, \epsilon_2, \ldots \epsilon_n$ that minimize $\max_j |X_j|$ where $X_j = \sum_{i=1}^n \epsilon_i v_{ij}$. Hoeffding tells us that choosing the ϵ_i independently at random gives a nonzero probability that this quantity is at most $\sqrt{2n \ln 2m}$.

Because there may be very complicated dependence between the X_j, it is difficult to calculate the probability of the event $\bigcup_j [|X_j| \geq t]$, whether

conditioned on some of the ϵ_i or not. However, we can calculate the probability of the individual events $[|X_j| \geq t]$ exactly. Conditioning on $\epsilon_1, \ldots, \epsilon_k$, the expected value of X_j is just $\ell = \sum_{i=1}^{k} \epsilon_i v_{ij}$, and the distribution of $Y = X_j - \mathrm{E}[X_j]$ is the sum $r \leq n - k$ independent ± 1 random variables. So

$$\begin{aligned}\Pr[|X_j| > t \mid \epsilon_1, \ldots, \epsilon_k] &= 1 - \Pr[-t - \ell \leq Y \leq t - \ell] \\ &= 1 - \sum_{i=\lceil \frac{-t-\ell+r}{2} \rceil}^{\lfloor \frac{t-\ell+r}{2} \rfloor} \binom{r}{i} 2^{-r}.\end{aligned}$$

This last expression involves a linear number of terms, each of which we can calculate using a linear number of operations on rational numbers that fit in a linear number of bits, so we can calculate the probability exactly in polynomial time by just adding them up.

For our pessimistic estimator, we take

$$U(\epsilon_1, \ldots, \epsilon_k) = \sum_{i=j}^{n} \Pr\left[|X_j| > \sqrt{2n \ln 2m} \;\middle|\; \epsilon_1, \ldots, \epsilon_k\right].$$

Since each term in the sum is a Doob martingale, the sum is a martingale as well, so $\mathrm{E}[U(\epsilon_1, \ldots, \epsilon_{k+1}) \mid \epsilon_1, \ldots, \epsilon_k] = U(\epsilon_1, \ldots, \epsilon_k)$. It follows that for any choice of $\epsilon_1, \ldots, \epsilon_k$ there exists some ϵ_{k+1} such that $U(\epsilon_1, \ldots, \epsilon_k) \geq U(\epsilon_1, \ldots, \epsilon_{k+1})$, and we can determine this winning ϵ_{k+1} explicitly. Our previous argument shows that $U(\langle\rangle) < 1$, which implies that our final value $U(\epsilon_1, \ldots, \epsilon_n)$ will also be less than 1. Since U is an integer, this means it must be 0, and we find an assignment in which $|X_j| < \sqrt{2n \ln 2m}$ for all j.

Chapter 15

Probabilistically-checkable proofs

In this chapter, we discuss the **PCP theorem** [ALM+98], which is an alternative characterization of **NP** that can be used to show hardness for various approximation problems assuming $\mathbf{P} \neq \mathbf{NP}$. What connects the PCP theorem to randomized algorithms is that class of **probabilistically-checkable proofs** that it describes are what you get when you replace the deterministic verifier that checks a witness in the usual definition of **NP** with a randomized algorithm. This allows for very efficient verification, since a randomized verified can demand a proof that includes sufficient internal consistency checks that a spot-check of only a few randomly-chosen bits of the proof is enough to detect an incorrect proof with constant probability. Formally, this means that we consider a verifier that is a probabilistic Turing machine that uses r random bits to select q bits of the proof to look at, and accepts bad proofs with less than some constant probability ρ while accepting all good proofs.

This turns out to have strong consequences for approximation algorithms. We can think of a probabilistically-checkable proof as a kind of constraint satisfaction problem, where the constraints apply to the tuples of q bits that the verifier might look at, the number of constraints is bounded by the number of possible random choices 2^r, and each constraint enforces some condition on those q bits corresponding to the verifier's response to seeing them. If we can satisfy at least $\rho \cdot 2^r$ of the constraints, we've constructed a proof that is not bad. This means that there is a winning certificate for our original **NP** machine, and that whatever input x we started with is in the language accepted by that machine. So a polynomial-time approximation algorithm

for this constraint satisfaction problem will let us generate probabilistically-checkable proofs for the corresponding problem in **NP**, which means there can't be such an approximation algorithm unless $\mathbf{P} = \mathbf{NP}$.

The above is only a sketchy, high-level overview of where we are going. Below we fill in some of the details. We are mostly following the approach of [AB07, §§18.1–18.4], with much of the actual text of this chapter adapted from [Asp20].

15.1 Probabilistically-checkable proofs

A $\langle r(n), q(n), \rho \rangle$-**PCP verifier** for a language L consists of two polynomial-time computable functions f and g, where:

- $f(x, r)$ takes an input x of length n and a string r of length $r(n)$ and outputs a sequence i of $q(n)$ indices $i_1, i_2, \ldots, i_{q(n)}$, each in the range $0 \ldots q(n) \cdot 2^{r(n)}$; and
- $g(x, \pi_i)$ takes as input the same x as f and a sequence $\pi_i = \pi_{i_1} \pi_{i_2} \ldots \pi_{i_{(q)n}}$ and outputs either 1 (for accept) or 0 for (reject); and
- if $x \in L$, then there exists a sequence π that causes $g(x, \pi_{f(x,r)})$ to output 1 always (**completeness**); and
- if $x \notin L$, then for any sequence π, $g(x, \pi_{f(x,r)})$ outputs 1 with probability at most ρ (**soundness**).

We call the string π a **probabilistically-checkable proof** or **PCP** for short.

Typically, ρ is set to $1/2$, and we just write $\langle r(n), q(n) \rangle$-**PCP** verifier for $\langle r(n), q(n), 1/2 \rangle$-**PCP** verifier.

The class $\mathbf{PCP}(r(n), q(n))$ is the class of languages L for which there exists an $\langle O(r(n)), O(q(n)), 1/2 \rangle$-**PCP** verifier.

The **PCP theorem** says that $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$. That is, any language in **NP** can be recognized by a **PCP**-verifier that is allowed to look at only a constant number of bits selected using $O(\log n)$ random bits from a proof of polynomial length, which is fooled at most half the time by bad proofs. In fact, 3 bits is enough [Hås01]. We won't actually prove this here, but we will describe some consequences of the theorem, and give some hints about how the proof works.

15.2 A PCP for GRAPH NON-ISOMORPHISM

The **GRAPH NON-ISOMORPHISM** (**GNI**) problem takes as input two graphs G and H, and asks if G is not isomorphic to H (written $G \not\simeq H$). Recall that G *is* isomorphic to H (written $G \simeq H$) if there is a permutation of the vertices of G that makes it equal to H. An **NP**-machine can easily solve the GRAPH ISOMORPHISM problem by guessing the right permutation, which puts the complementary GRAPH NON-ISOMORPHISM problem in **coNP**.[1] But in general we don't know if problems in **coNP**problems have **NP**-style proofs, and there is no obvious general method that a prover could use to convince a deterministic verifier that two graphs are *not* isomorphic.

Instead, we'll describe two protocols that allow an omniscient prover to demonstrate that two graphs are not isomorphic to a suspicious, randomized verifier. The first, described in §15.2.1, is an **interactive proof**, meaning that the verifier can ask questions that the prover must answer. The second, described in §15.2.2, is not interactive: instead, the prover supplies a (very large) proof that the verifier can check by looking at a single randomly-chosen bit.

15.2.1 GRAPH NON-ISOMORPHISM with private coins

Here is how the prover can convince a verifier that two graphs are not isomorphic, assuming that the verifier can flip coins that are not visible to the prover.

Given graphs G and H, the verifier picks one or the other with equal probability, then randomly permutes its vertices to get a test graph T. It then asks the prover which of G or H it picked.

If the graphs are not isomorphic, the prover can distinguish $T \simeq G$ from $T \simeq H$ and answer the question correctly with probability 1. If they are isomorphic, then the prover can only guess: T gives no information at all about which of G or H was used to generate it. So in this case it answers correctly only with probability 1/2. This gives us the completeness and soundness properties needed for a probabilistically-checkable proof. Unfortunately we need to do more work to make this non-interactive.

[1] Neither problem is known not to be in **P**, even under the assumption that $\mathbf{P} \neq \mathbf{NP}$. The exact complexity of GRAPH ISOMORPHISM is still open.

15.2.2 A probabilistically-checkable proof for GRAPH NON-ISOMORPHISM

To turn the interactive proof into a probabilistically-checkable proof, have the prover build a bit-vector π indexed by every possible graph on n vertices, writing a 1 for each graph that is isomorphic to H. Now the verifier can use $\Theta(n^2)$ random bits to construct T as above, look it up in this gigantic table, and accept if and only if (a) it chose G and $\pi[T] = 0$, or (b) it chose H and $\pi[T] = 1$. If G and H are non-isomorphic, the verifier accepts every time. But if they are isomorphic, no matter what proof π' is supplied, there is at least a $1/2$ chance that $\pi'[T]$ is wrong. This puts GRAPH NON-ISOMORPHISM in $\mathbf{PCP}(n^2, 1)$.

(A minor technical issue here is that the verifier only needs $O(n \log n)$ bits to construct T. But we give it $\Theta(n^2)$ bits, so that the table can be big enough to hold all graphs and still fit in the maximum proof size $q(n)2^{r(n)}$.)

15.3 $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly}(n), 1)$

Here we give a weak version of the PCP theorem, adapted from [AB07, §18.4], showing that any problem in **NP** has a probabilistically-checkable proof where the verifier uses polynomially-many random bits but only needs to look at a constant number of bits of the proof, which we can state succinctly as $\mathbf{NP} \subseteq \mathbf{PCP}(\text{poly}(n), 1)$.[2] The proof itself will be exponentially long.

The central step is to construct a $\langle \text{poly}(n), 1 \rangle$-PCP for a particular **NP**-complete problem; we can then take any other problem in **NP**, reduce it to this problem, and use the construction to get a PCP for that problem as well.

15.3.1 QUADEQ

The particular problem we will look at is QUADEQ, the language of systems of quadratic equations over $\mathbb{Z}_2$ that have solutions.

This is in **NP** because we can guess and verify a solution; it's **NP**-hard because we can use quadratic equations over $\mathbb{Z}_2$ to encode instances of SAT, using the representation 0 for false, 1 for true, $1 - x$ for $\neg x$, xy for $x \wedge y$, and $1 - (1 - x)(1 - y) = x + y + xy$ for $x \vee y$. We may also need to introduce

[2] This is a rather weak result, since (a) the full PCP theorem gives **NP** using only $O(\log n)$ random bits, and (b) $\mathbf{PCP}(\text{poly}(n), 1)$ is known to be equal to **NEXP** [BFL91]. But the construction is still useful for illustrating many of the ideas behind probabilistically-checkable proofs.

auxiliary variables to keep the degree from going up: for example, to encode the clause $x \vee y \vee z$, we introduce an auxiliary variable q representing $x \vee y$ and enforce the constraints $q = x \vee y$ and $1 = q \vee z = x \vee y \vee z$ using two equations

$$x + y + xy = q,$$
$$q + z + qz = 1.$$

It will be helpful later to rewrite these in a standard form with only zeros on the right:

$$q + x + y + xy = 0$$
$$q + z + qz + 1 = 0.$$

This works because we can move summands freely from one side of an equation to the other since all addition is mod 2.

15.3.2 The Walsh-Hadamard Code

An **NP** proof for QUADEQ just gives an assignment to the variables that makes all the equations true. Unfortunately, this requires looking at the entire proof to check it. To turn this into a $\mathbf{PCP}(\text{poly}(n), 1)$ proof, we will make heavy use of a rather magical **error-correcting code** called the **Walsh-Hadamard code**.

This code expands an n-bit string x into a 2^n-bit **codeword** $H(x)$, where $H(x)_i = x \cdot i$ when x and the index i are both interpreted as n-dimensional vectors over $\mathbb{Z}_2$ and $\cdot$ is the usual vector dot-product $\sum_{i=1}^{n} x_j i_j$. This encoding has several very nice properties, all of which we will need:

1. It is a linear code: $H(x + y) = H(x) + H(y)$ when all strings are interpreted as vectors of the appropriate dimension over $\mathbb{Z}_2$.

2. It is an error-correcting code with distance 2^{n-1}. If $x \neq y$, then exactly half of all i will give $x \cdot i \neq y \cdot i$. This follows from the **subset sum principle**, which says that a random subset of a non-empty set S is equally likely to have an even or odd number of elements.[3] So for any particular nonzero x, exactly half of the $x \cdot i$ values will be

[3] Proof: If S is non-empty, pick some element x, and then the function f that removes x if present and adds x if missing is a bijection between even and odd subsets. If you like generating functions, an alternative proof uses the Binomial Theorem to show $\sum_{\text{even } i} \binom{n}{i} - \sum_{\text{odd } i} \binom{n}{i} = \sum_{i=0}^{n} (-1)^n \binom{n}{i} = (1 + (-1))^n = 0^n = 0$ when $n \neq 0$.

1, since i includes each one in x with independent probability 1/2. This makes $d(H(0), H(x)) = 2^{n-1}$. But then the linearity of H gives $d(H(x), H(y)) = d(H(0), d(H(x+y))) = 2^{n-1}$ whenever $x \neq y$.

3. It is **locally testable**: Given an alleged codeword w, we can check if w is close to being a legitimate codeword $H(x)$ by sampling a constant number of bits from w. (We do not need to know what x is to do this.)

   Our test is: Pick two indices i and j uniformly at random, and check if $w_i + w_j = w_{i+j}$. A legitimate codeword will pass this test always. It is also possible to show using Fourier analysis (see [AB07, Theorem 19.9]) that if w passes this test with probability $\rho \geq 1/2$, then there is some x such that $\Pr_i[H(x)_i = w_i] \geq \rho$ (equivalently, $d(H(x), w) \leq 2^n(1-\rho)$, in which case we say w is **ρ-close** to $H(x)$.

4. It is **locally decodable**: If w is ρ-close to $H(x)$, then we can compute $H(x)_i$ by choosing a random index r and computing $w_r + w_{i+r}$. This will be equal to $H(x)_i$ if both bits are correct (by linearity of H). The probability that both bits are correct is at least $1 - 2\delta$ if $\rho = 1 - \delta$.

5. It allows us to check an unlimited number of linear equations in x by looking up a single bit of $H(x)$. This again uses the subset sum principle. Give a system of linear equations $x \cdot y_1 = 0, x \cdot y_2 = 0, \ldots x \cdot y_m = 0$, choose a random subset S of $\{1, \ldots, m\}$, let $i = \sum_{j \in S} y_j$, and query $H(x)_i = x \cdot \sum_{j \in S} y_j$. This will be 0 always if the equations hold and 1 with probability 1/2 if at least one is violated.

   This gets a little more complicated if we have any ones on the right-hand side. But we can handle an equation of the form $x \cdot y = 1$ by rewriting it as $x \cdot y + 1 = 0$, and then extending x to include an extra constant 1 bit (which we can test is really one by looking up $H(x)_i$ for an appropriate index i).

15.3.3 A PCP for QUADEQ

So now to construct a PCP for QUADEQ, we build:

1. An n-bit solution u to the system of quadratic equations, which we think of as a function $f : \{0,1\}^n \to \{0,1\}$ and encode as $f = H(u)$.

2. An n^2-bit vector $w = u \otimes u$ where $(u \otimes u)_{ij} = u_i u_j$, which we encode as $g = H(u \otimes u)$.

To simplify our life, we will assume that one of the equations is $x_1 = 1$ so that we can use this constant 1 later (note that we can trivially reduce the unrestricted version of QUADEQ to the version that includes this assumption by adding an extra variable and equation). A different approach that does not require this assumption is given in [AB07, §18.4.2, Step 3].

To test this PCP, the verifier checks:

1. That f and g are $(1 - \delta)$-close to real codewords for some suitably small δ.

2. That for some random r, s, $f(r)f(s) = g(r \otimes s)$. This may let us know if w is inconsistent with u. Define W as the $n \times n$ matrix with $W_{ij} = w_{ij}$ and U as the $n \times n$ matrix $U = u \otimes u$ (so $U_{ij} = u_i u_j$). Then $g(r \otimes s) = w \cdot (r \otimes s) = \sum_{ij} w_{ij} r_i s_j = rWs$ and $f(r)f(s) = (u \cdot r)(u \cdot s) = (\sum_i u_i r_i)\left(\sum_j u_j s_j\right) = \sum_{ij} r_i U_{ij} r_j = rUs$, where we are treating r as a row vector and s as a column vector. Now apply the random subset principle to argue that if $U \neq W$, then $rU \neq rW$ at least half the time, and if $rU \neq rW$, then $rUs \neq rWs$ at least half the time. This gives a probability of at least $1/4$ that we catch $U \neq W$, and we can repeat the test a few times to amplify this to whatever constant we want.

3. That our extra constant-1 variable is in fact 1 (lookup on u).

4. That w encodes a satisfying assignment for the original problem. This just involves checking a system of linear equations using w.

Since we can make each step fail with only a small constant probability, we can make the entire process fail with the sum of these probabilities, also a small constant.

15.4 PCP and approximability

Suppose we want to use the full **PCP** theorem $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$ to actually decide some language L in **NP**. What do we need to do?

15.4.1 Approximating the number of satisfied verifier queries

If we can somehow find a PCP for $x \in L$ and verify it, then we know $x \in L$. So the obvious thing is to try to build an algorithm for generating PCPs. But actually generating a PCP may be hard. Fortunately, even getting an good approximation will be enough. We illustrate the basic idea using MAX

SAT, the problem of finding an assignment that maximizes the number of satisfied clauses in a 3CNF formula.

Suppose that we have some language L with a PCP verifier V. If $x \in L$, there exists a proof of polynomial length such that every choice of $q = O(\log n)$ bits by V from the proof will be accepted by V. We can encode this verification step as a Boolean formula: for each set of bits $S = \{i_1, \ldots, i_q\}$, write a polynomial-size formula ϕ_S with variables in π that checks if V will accept $\pi_{i_1}, \ldots, \pi_{i_q}$ for our given input x. Then we can test if $x \in L$ by testing if $\phi = \bigwedge_S \phi_S$ is satisfiable or not.

But we can do better than this. Suppose that we can approximate the number of ϕ_S that can be satisfied to within a factor of $2 - \epsilon$. Then if ϕ has an assignment that makes all the ϕ_S true (which followed from completeness if $x \in L$), our approximation algorithm will give us an assignment that makes at least a $\frac{1}{2-\epsilon} > \frac{1}{2}$ fraction of the ϕ_S true. But we can never make more than $\frac{1}{2}$ of the ϕ_S true if $x \notin L$. So we can run our hypothetical approximation algorithm, and if it gives us an assignment that satisfies more than half of the ϕ_S, we know $x \in L$. If the approximation runs in **P**, we just solved SAT in **P** and showed **P** = **NP**.

15.4.2 Gap-preserving reduction to MAX SAT

Maximizing the number of subformulas ϕ_S that are satisfied is a strange problem, and we'd like to state this result in terms of a more traditional problem like MAX SAT. We can do this by converting each ϕ_S into a 3CNF formula (which makes ϕ also 3CNF), but the cost is that we reduce the **gap** between negative instances $x \notin L$ and negative instances $x \in L$.

The **PCP** theorem gives us a gap of $(1/2, 1)$ between negative and positive instances. If each ϕ_S is represented by k 3CNF clauses, then it may be that violating a single ϕ_S only maps to violating one of those k clauses. So where previously we either satisfied at most $1/2$ of the ϕ_S or all of them, now we might have a negative instance where we can still satisfy a $1 - \frac{1}{2k}$ fraction of the clauses. So we only get **P** = **NP** if we are given a poly-time approximation algorithm for MAX SAT that is at least this good; or, conversely, we only show that **P** $\neq$ **NP** implies that there is no MAX SAT approximation that gets more than $1 - \frac{1}{2k}$ of optimal.

This suggests that we want to find a version of the **PCP** theorem that makes k as small as possible. Fortunately, Håstad [Hås01] showed that it is possible to construct a PCP-verifier for 3SAT with the miraculous property that (a) q is only 3, and (b) the verification step involves testing only if $\pi_{i_1} \oplus \pi_{i_2} \oplus \pi_{i_3} = b$, where i_1, i_2, i_3, and b are generated from the random bits.

There is a slight cost: the completeness parameter of this verifier is only $1 - \epsilon$ for any fixed $\epsilon > 0$, meaning that it doesn't always recognize a valid proof, and the soundness parameter is $1/2+\epsilon$. But checking $\pi_{i_1} \oplus \pi_{i_2} \oplus \pi_{i_3} = b$ requires a 3CNF formula of only 4 clauses. So this means that there is no approximation algorithm for MAX SAT that does better than $7/8 + \delta$ of optimal in all cases, unless $\mathbf{P} = \mathbf{NP}$. This matches the 7/8 upper bound given by just picking an assignment at random.[4]

This is an example of a reduction argument, since we reduced 3SAT first to a problem of finding a proof that would make a particular PCP-verifier happy and then to MAX SAT. The second reduction is an example of a **gap-preserving reduction**, in that it takes an instance of a problem with a non-trivial gap $(1/2 + \epsilon, 1 - \epsilon)$ and turns it into an instance of a problem with a non-trivial gap $(7/8 + \epsilon, 1 - \epsilon)$. Note that do be gap-preserving, a reduction doesn't have to preserve the value of the gap, it just has to preserve the existence of a gap. So a **gap-reducing reduction** like this is still gap-preserving. We can also consider **gap-amplifying reductions**: in a sense, Håstad's verifier gives a reduction from 3SAT to 3SAT that amplifies the reduction from the trivial $(1 - 1/m, 1)$ that follows from only being able to satisfy $m - 1$ of the m clauses in a negative instance to the much more useful $(1/2 + \epsilon, 1 - \epsilon)$.

15.4.3 Other inapproximable problems

Using inapproximability of MAX SAT, we can find similar inapproximability results for other **NP**-complete optimization problems by looking for gap-preserving reductions from MAX SAT. In many cases, we can just use whatever reduction we already had for showing that the target problem was **NP**-hard. This gives constant-factor inapproximability bounds for problems like GRAPH 3-COLORABILITY (where the value of a solution is the proportion of two-colored edges) and MAXIMUM INDEPENDENT SET (where the value of a solution is the size of the independent set). In each

[4] It's common in the approximation-algorithm literature to quote approximation ratios for maximization problems as the fraction of the best solution that we can achieve, as in a 7/8-approximation for MAX SAT satisfying 7/8 of the maximum possible clauses. This leads to rather odd statements when we start talking about lower bounds ("you can't do better than $7/8 + \delta$") and upper bounds ("you can get at least 7/8"), since the naming of the bounds is reversed from what they actually say. For this reason complexity theorists have generally standardized on always treating approximation ratios as greater than 1, which for maximization problems means reporting the inverse ratio, $8/7 - \epsilon$ in this case. I like 7/8 better than 8/7, and there is no real possibility of confusion, so I will stick with 7/8.

case we observe that a partial solution to the target problem maps back to a partial solution to the original SAT problem.

In some cases we can do better, by applying a gap-amplification step. For example, suppose that no polynomial-time algorithm for INDEPENDENT SET can guarantee an approximation ratio better than ρ, assuming $\mathbf{P} \neq \mathbf{NP}$. Given a graph G, construct the graph G^k on $\binom{n}{k}$ vertices where each vertex in G^k represents a set of k vertices in G, and ST is an edge in G^k if $S \cup T$ is *not* an independent set in G. Let I be an independent set for G. Then the set I^k of all k-subsets of I is an independent set in G^k ($S \cup T \subseteq I$ is an independent set for any S and T in I^k). Conversely, given any independent set $J \subseteq G^k$, its union $\bigcup J$ is an independent set in G (because otherwise there is an edge either within some element of J or between two elements of J). So any maximum independent set in G^k will be I^k for some maximum independent set in G.

This amplifies approximation ratios: given an independent set I such that $|I|/|OPT| = \rho$, then $|I^k|/|OPT^k| = \binom{|I|}{k}/\binom{|OPT|}{k} \approx \rho^k$. If k is constant, we can compute G^k in polynomial time. If we can then compute a ρ^k-approximation to the maximum independent set in G^k, we can take the union of its elements to get a ρ-approximation to the maximum independent set in G. By making k sufficiently large, this shows that approximating the maximum independent set to within any constant $\epsilon > 0$ is **NP**-hard.

There is a stronger version of this argument that uses expander graphs to get better amplification, which shows that $n^{-\delta}$ approximations are also **NP**-hard for any $\delta < 1$. See [AB07, §18.3] for a description of this argument.

15.5 Dinur's proof of the PCP theorem

Here we give a very brief outline of Dinur's proof of the **PCP** theorem [Din07]. This is currently the simplest known proof of the theorem, although it is still too involved to present in detail here. For a more complete description, see §18.5 of [AB07], or Dinur's paper, which is pretty accessible.

A **constraint graph** is a graph $G = (V, E)$, where the vertices in V are interpreted as variables, and each edge uv in E carries a **constraint** $c_{uv} \subseteq \Sigma^2$ that specifies what assignments of values in some **alphabet** Σ are permitted for the endpoints of uv. A **constraint satisfaction problem** asks for an assignment $\sigma : V \to \Sigma$ that minimizes $\mathrm{UNSAT}_\sigma(G) = \Pr_{uv \in E}[\langle \sigma(u), \sigma(v) \rangle \notin c_{uv}]$, the probability that a randomly-chosen constraint is unsatisfied. The quantity $\mathrm{UNSAT}(G)$ is defined as minimum value of $\mathrm{UNSAT}_\sigma(G)$: this is the smallest proportion of constraints that we must leave unsatisfied. In

the other direction, the **value** $\text{val}(G)$ of a constraint satisfaction problem is $1 - \text{UNSAT}(G)$: this is the largest proportion of constraints that we can satisfy. [5]

An example of a constraint satisfaction problem is GRAPH 3-COLORABILITY: here $\Sigma = \{r, g, b\}$, and the constraints just require that each edge on the graph we are trying to color (which will be the same as the constraint graph G!) has two different colors on its endpoints. If a graph G with m edges has a 3-coloring, then $\text{UNSAT}(G) = 0$ and $\text{val}(G) = 1$; if G does not, then $\text{UNSAT}(G) \geq 1/m$ and $\text{val}(G) \leq 1 - 1/m$. This gives a $(1 - 1/m, 1)$ gap between the best value we can get for non-3-colorable vs. 3-colorable graphs. Dinur's proof works by amplifying this gap.

Here is the basic idea:

1. We first assume that our input graph is k-regular (all vertices have the same degree k) and an expander (every subset S with $|S| \leq m/2$ has $\delta|S|$ external neighbors for some constant $\delta > 0$). Dinur shows that even when restricted to graphs satisfying these assumptions, GRAPH 3-COLORABILITY is still **NP**-hard.

2. We then observe that coloring our original graph G has a gap of $(1-1/m, 1)$, or that $\text{UNSAT}(G) \geq 1/m$. This follows immediately from the fact that a bad coloring must include at least one monochromatic edge.

3. To amplify this gap, we apply a two-stage process.

   First, we construct a new constraint graph G' (that is no longer a graph coloring problem) with n vertices, where the constraint graph has an edge between any two vertices at distance $2d + 1$ or less in G, the label on each vertex v is a "neighborhood map" assigning a color of every vertex within distance d of v, and the constraint on each edge uv says that the maps for u and v (a) assign the same color to each

[5] Though Dinur's proof doesn't need this, we can also consider a **constraint hypergraph**, where each q-**hyperedge** is a q-tuple $e = v_1 v_2 \dots v_q$ relating q vertices, and a constraint c_e is a subset of Σ^q describing what assignments are permitted to the vertices in e. As in the graph case, our goal is to minimize $\text{UNSAT}_\sigma(G)$, which is now the proportion of hyperedges whose constraints are violated, or equivalently to maximize $\text{val}_\sigma(G) = 1 - \text{UNSAT}_\sigma(G)$. This gives us the q-CSP problem: given a q-ary constraint hypergraph, find an assignment σ that maximizes $\text{val}_\sigma(G)$. An example of a constraint hypergraph arises from 3SAT. Given a formula ϕ, construct a graph G_ϕ in which each vertices represents a variable, and the constraint on each 3-hyperedge enforces least one of the literals in some clause is true. The MAX 3SAT problem asks to find an assignment σ that maximizes the proportion of satisfied clauses, or $\text{val}_\sigma(G_\phi)$.

vertex in the overlap between the two neighborhoods, and (b) assigns colors to the endpoints of any edge in either neighborhood that are permitted by the constraint on that edge. Intuitively, this means that a bad edge in a coloring of G will turn into many bad edges in G', and the expander assumption means that many bad edges in G will also turn into many bad edges in G'. In particular, Dinur shows that with appropriate tuning this process amplifies the UNSAT value of G by a constant. Unfortunately, we also blow up the size of the alphabet by $\Theta(k^d)$.

So the second part of the amplification knocks the size of the alphabet back down to 2. This requires replacing each node in G' with a set of nodes in a new constraint graph G'', where the state of the nodes in the set encodes the state of the original node, and some coding-theory magic is used to preserve the increased gap from the first stage (we lose a little bit, but not as much as we gained).

The net effect of both stages is to take a constraint graph G of size n with $\text{UNSAT}(G) \geq \epsilon$ and turn it into a constraint graph G'' of size cn, for some constant c, with $\text{UNSAT}(G'') \geq 2\epsilon$.

4. Finally, we repeat this process $\Theta(\log m) = \Theta(\log n)$ times to construct a constraint graph with size $c^{O(\log n)}n = \text{poly}(n)$ and gap $(1/2, 1)$. Any solution to this constraint graph gives a PCP for GRAPH 3-COLORABILITY for the original graph G.

15.6 The Unique Games Conjecture

The **PCP** theorem, assuming $\mathbf{P} \neq \mathbf{NP}$, gives us fairly strong inapproximability results for many classic optimization problems, but in many cases these are not tight: there is a still a gap between the lower bound and the best known upper bound. The **Unique Games Conjecture** of Khot [Kho02], if true, makes many of these bounds tight.

The Unique Games Conjecture was originally formulated in terms of an interactive proof game with two provers. In this game, the verifier V picks a query q_1 to ask of prover P_1 and a query q_2 to ask of prover P_2, and then checks consistency of the prover's answers a_1 and a_2. (The provers cannot communicate, and so answer the queries independently, although they can coordinate their strategy before seeing the queries.) This gives a **unique game** if, for each answer a_1 there is exactly one answer a_2 that will cause V to accept.

Equivalently, we can model a unique game as a restricted 2-CSP: the labels on the vertices are the answers, and the consistency condition on each edge is a bijection between the possible labels on each endpoint. This corresponds to the two-prover game the same way PCPs correspond to single-prover games, in that a labeling just encodes the answers given by each prover.

A nice feature of unique games is that they are easy to solve in polynomial time: pick a label for some vertex, and then use the unique constraints to deduce the labels for all the other vertices. So the problem becomes interesting mostly when we have games for which there is no exact solution.

For the Unique Games Conjecture, we consider the set of all unique games with gap $(\delta, 1-\epsilon)$; that is, the set consisting of the union of all unique games with approximations with ratio $1-\epsilon$ or better and all unique games with no approximations with ratio better than δ. The conjecture states that for any δ and ϵ, there exists some alphabet size k such that it is **NP**-hard to determine of these two piles a unique game G with this alphabet size lands in.[6]

Unfortunately, even though the Unique Games Conjecture has many consequences that are easy to state (for example, the usual 2-approximation to MINIMUM VERTEX COVER is optimal, as is the 0.878-approximation to MAX CUT of Goemans and Williamson [GW95]), actually proving these consequences requires fairly sophisticated arguments. So we won't attempt to do any of them here, and instead will point the interested reader to Khot's 2010 survey paper [Kho10], which gives a table of bounds known at that time and citations to where they came from.

There is no particular consensus among complexity theorists as to whether the Unique Games Conjecture is true or not, but it would be nifty if it were.

[6] Note that this is not a decision problem, in that the machine M considering G does not need to do anything sensible if G is in the gap; instead, it is an example of a **promise problem** where we have two sets L_0 and L_1, $M(x)$ must output i when $x \in L_i$, but $L_0 \cup L_1$ does not necessarily cover all of $\{0,1\}^*$.

Chapter 16

Quantum computing

Quantum computing is a currently almost-entirely-theoretical branch of randomized algorithms that attempts to exploit the fact that probabilities at a microscopic scale arise in a mysterious way from more fundamental **probability amplitudes**, which are complex-valued and can cancel each other out where probabilities can only add. In a quantum computation, we replace random bits with **quantum bits**—**qubits** for short—and replace random updates to the bits with quantum operations.

To explain how this works, we'll start by re-casting our usual model of a randomized computation to make it look more like the standard **quantum circuits** of Deutsch [Deu89]. We'll then get quantum computation by replacing all the real-valued probabilities with complex-valued amplitudes.

16.1 Random circuits

Let's consider a very simple randomized computer whose memory consists of two bits. We can describe our knowledge of the state of this machine using a vector of length 4, with the coordinates in the vector giving the probability of states 00, 01, 10, and 11. For example, if we know that the initial state is always 00, we would have the (column) vector

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Any such **state vector** x for the system must consist of non-negative real values that sum to 1; this is just the usual requirement for a discrete

probability space. Operations on the state consist of taking the old values of one or both bits and replacing them with new values, possibly involving both randomness and dependence on the old values. The law of total probability applies here, so we can calculate the new state vector x' from the old state vector x by the rule

$$x'_{b'_1 b'_2} = \sum_{b_1 b_2} x_{b_1 b_2} \Pr\left[X_{t+1} = b'_1 b'_2 \mid X_t = b_1 b_2\right].$$

These are linear functions of the previous state vector, so we can summarize the effect of our operation using a transition matrix A, where $x' = Ax$.[1]

We imagine that these operations are carried out by feeding the initial state into some circuit that generates the new state. This justifies the calling this model of computation a **random circuit**. But the actual implementation might be an ordinary computer than is just flipping coins. If we can interpret each step of the computation as applying a transition matrix, the actual implementation doesn't matter.

For example, if we negate the second bit 2/3 of the time while leaving the first bit alone, we get the matrix

$$A = \begin{bmatrix} 1/3 & 2/3 & 0 & 0 \\ 2/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 2/3 \\ 0 & 0 & 2/3 & 1/3 \end{bmatrix}.$$

One way to derive this matrix other than computing each entry directly is that it is the **tensor product** of the matrices that represent the operations on the individual bits. The idea here is that the tensor product of A and B, written $A \otimes B$, is the matrix C with $C_{ij,k\ell} = A_{ik} B_{j\ell}$. We're cheating a little bit by allowing the C matrix to have indices consisting of pairs of indices, one for each of A and B; there are more formal definitions that justify this at the cost of being harder to understand.

In this particular case, we have

$$\begin{bmatrix} 1/3 & 2/3 & 0 & 0 \\ 2/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 2/3 \\ 0 & 0 & 2/3 & 1/3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix}.$$

[1] Note that this is the reverse of the convention we adopted for Markov chains in Chapter 10. There it was convenient to have $P_{ij} = p_{ij} = \Pr[X_{t+1} = j \mid X_t = i]$. Here we defer to the physicists and make the update operator come in front of its argument, like any other function.

The first matrix in the tensor product gives the update rule for the first bit (the identity matrix—do nothing), while the second gives the update rule for the second.

Some operations are not decomposable in this way. If we swap the values of the two bits, we get the matrix

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

which maps 00 and 11 to themselves but maps 01 to 10 and vice versa.

The requirement for all of these matrices is that they be **stochastic**. This means that each column has to sum to 1, or equivalently that $1A = 1$, where 1 is the all-ones vector. This just says that our operations map probability distributions to probability distributions; we don't apply an operation and find that the sum of our probabilities is now 3/2 or something. (Proof: If $1A = 1$, then $|Ax|_1 = 1(Ax) = (1A)x = 1x = |x|_1$.)

A **randomized computation** in this model now consists of a sequence of these stochastic updates to our random bits, and at the end performing a **measurement** by looking at what the values of the bits actually are. If we want to be mystical about it, we could claim that this measurement collapses a probability distribution over states into a single unique state, but really we are just opening the box to see what we got.

For example, we could generate two bias-2/3 coin-flips by starting with 00 and using the algorithm flip second, swap bits, flip second, or in matrix terms:

$$\begin{aligned} x_{\text{out}} &= ASAx_{\text{in}} \\ &= \begin{bmatrix} 1/3 & 2/3 & 0 & 0 \\ 2/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 2/3 \\ 0 & 0 & 2/3 & 1/3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/3 & 2/3 & 0 & 0 \\ 2/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 2/3 \\ 0 & 0 & 2/3 & 1/3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1/9 \\ 2/9 \\ 2/9 \\ 4/9 \end{bmatrix}. \end{aligned}$$

When we look at the output, we find 11 with probability 4/9, as we'd expect, and similarly for the other values.

16.2 Bra-ket notation

A notational oddity that scares many people away from quantum mechanics in general and quantum computing in particular is the habit of practitioners in these fields of writing basis vectors and their duals using **bra-ket notation**, a kind of typographical pun invented by the physicist Paul Dirac [Dir39].

This is based on a traditional way of writing an **inner product** of two vectors x and y in **bracket form** as $\langle x|y\rangle$. The interpretation of this is $\langle x|y\rangle = x^*y$, where x^* is the **conjugate transpose** of x.

For real-valued x the conjugate transpose is the same as the transpose. For complex-valued x, each coordinate $x_i = a+bi$ is replaced by its **complex conjugate** $\bar{x}_i = a - bi$.) Using the conjugate transpose makes $\langle x|x\rangle$ equal $|x|_2^2$ when x is complex-valued.

For example, for our vector x_{in} above that puts all of its probability on 00, we have

$$\langle x_{\text{in}}|x_{\text{in}}\rangle = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = 1. \tag{16.2.1}$$

The typographic trick is to split in half both $\langle x|y\rangle$ and its expansion. For example, we could split (16.2.1) as

$$\langle x_{\text{in}}| = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix} \qquad |x_{\text{in}}\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}.$$

In general, wherever we used to have a bracket $\langle x|y\rangle$, we now have a **bra** $\langle x|$ and a **ket** $|y\rangle$. These are just row vectors and column vectors, and $\langle x|$ is always the conjugate transpose of $|x\rangle$.

The second trick in bra-ket notation is to make the contents of the bra or ket an arbitrary name. For kets, this will usually describe some state. As an example, we might write x_{in} as $|00\rangle$ to indicate that it's the basis vector that puts all of its weight on the state 00. For bras, this is the linear operator that returns 1 when applied to the given state and 0 when applied to any orthogonal state. So $\langle 00|\,|00\rangle = \langle 00|00\rangle = 1$ but $\langle 00|\,|01\rangle = \langle 00|01\rangle = 0$.

16.2.1 States as kets

This notation is useful for the same reason that variable names are useful. It's much easier to remember that $|01\rangle$ refers to the distribution assigning

probability 1 to state 01 than it is to remember that $\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}^\top$ does.

Other vectors can be expressed using a linear combination of kets. For example, we can write

$$x_{\text{out}} = \frac{1}{9}|00\rangle + \frac{2}{9}|01\rangle + \frac{2}{9}|10\rangle + \frac{4}{9}|11\rangle .$$

This is not as compact as just writing out the vector as a matrix, but it has the advantage of clearly labeling what states the probabilities apply to.

16.2.2 Composition of kets

We'll interpret multiplication of kets as tensor product. For example:

$$|0\rangle|1\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |01\rangle .$$

This gives a way of composing states together. The general rule is $|x\rangle|y\rangle = |xy\rangle$. Because this is tensor product underneath, this is a linear operation, and it associates with other products in the obvious way.

16.2.3 Operators as sums of kets times bras

A similar trick can be used to express operators, like the swap operator S. We can represent S as a combination of maps from specific states to specific other states. For example, the operator

$$|01\rangle\langle 10| = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{bmatrix}$$

maps $|10\rangle$ to $|01\rangle$ (Proof: $|01\rangle\langle 10||10\rangle = |01\rangle\langle 10|10\rangle = |01\rangle$) and sends all other states to 0. Add up four of these mappings to get

$$S = |00\rangle\langle 00| + |10\rangle\langle 01| + |01\rangle\langle 10| + |11\rangle\langle 11| = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix} .$$

Here the bra-ket notation both labels what we are doing and saves writing a lot of zeros.

The intuition is that just like a ket represents a state, a bra represents a test for being in that state. So something like $|01\rangle\langle 10|$ tests if we are in the 10 state, and if so, sends us to the 01 state.

16.3 Quantum circuits

So how do we turn our random circuits into **quantum circuits**?

The first step is to replace our random bits with quantum bits (qubits).

For a single random bit, the state vector represents a probability distribution

$$p_0 |0\rangle + p_1 |1\rangle ,$$

where p_0 and p_1 are non-negative real numbers with $p_0 + p_1 = 1$.

For a single qubit, the state vector represents amplitudes

$$a_0 |0\rangle + a_1 |1\rangle ,$$

where a_0 and a_1 are complex numbers with $|a_0|^2 + |a_1|^2 = 1$.[2] The reason for this restriction on amplitudes is that if we measure a qubit, we will see state 0 with probability $|a_0|^2$ and state 1 with probability $|a_1|^2$. Unlike with random bits, these probabilities are not mere expressions of our ignorance but arise through a still somewhat mysterious process from the more fundamental amplitudes.[3]

[2] The **absolute value**, **norm**, or **magnitude** $|a + bi|$ of a complex number is given by $\sqrt{a^2 + b^2}$. When $b = 0$, this is the same as the absolute value for the corresponding real number. For any complex number x, the norm can also be written as $\sqrt{\bar{x}x}$, where $\bar{x}$ is the complex conjugate of x. This is because $\sqrt{(a + bi)(a - bi)} = \sqrt{a^2 - (bi)^2} = \sqrt{a^2 + b^2}$. The appearance of the complex conjugate here explains why we define $\langle x|y\rangle = x^* y$; the conjugate transpose means that for $\langle x|x\rangle$, when we multiply x_i^* by x_i we are computing a squared norm.

[3] In the old days of "shut up and calculate," this process was thought to involve the unexplained power of a conscious observer to collapse a superposition into a classical state. Nowadays the most favored explanation involves **decoherence**, the difficulty of maintaining superpositions in systems that are interacting with large, warm objects with lots of thermodynamic degrees of freedom (measuring instruments, brains). The decoherence explanation is particularly useful for explaining why real-world quantum computers have a hard time keeping their qubits mixed even when nobody is looking at them. Decoherence by itself does not explain *which* basis states a system collapses to. Since bases in linear algebra are pretty much arbitrary, it would seem that we could end up running into a physics version of Goodman's grue-bleen paradox [Goo83], but there are apparently ways of dealing with this too using a mechanism called **einselection** [Zur03] that favors classical states over weird ones. Since all of this is (a) well beyond my own limited comprehension of quantum mechanics and (b) irrelevant to the theoretical model we are using, these issues will not be discussed further.

With multiple bits, we get amplitudes for all combinations of the bits, e.g.

$$\frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right)$$

gives a state vector in which each possible measurement will be observed with equal probability $\left(\frac{1}{2}\right)^2 = \frac{1}{4}$. We could also write this state vector as

$$\begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}.$$

16.3.1 Quantum operations

In the random circuit model, at each step we pick a small number of random bits, apply a stochastic transformation to them, and replace their values with the results. In the quantum circuit model, we do the same thing, but now our transformations must have the property of being **unitary**. Just as a stochastic matrix preserves the property that the probabilities in a state vector sum to 1, a **unitary matrix** preserves the property that the squared norms of the amplitudes in a state vector sum to 1.

Formally, a square, complex matrix A is unitary if it preserves inner products: $\langle Ax|Ay\rangle = \langle x|y\rangle$ for all x and y. Alternatively, A is unitary if $A^*A = AA^* = I$, where A^* is the conjugate transpose of A, because if this holds, $\langle Ax|Ay\rangle = (Ax)^*(Ay) = x^*A^*Ay = x^*Iy = x^*y = \langle x|y\rangle$. Yet another way to state this is that the columns of A form an orthonormal basis: this means that $\langle A_i|A_j\rangle = 0$ if $i \neq j$ and 1 if $i = j$, which is just a more verbose way to say $A^*A = I$. The same thing also works if we consider rows instead of columns.

The rule then is: at each step, we can operate on some constant number of qubits by applying a unitary transformation to them. In principle, this could be *any* unitary transformation, but some particular transformations show up often in actual quantum algorithms.[4]

The simplest unitary transformations are permutations on states (like the operator that swaps two qubits), and rotations of a single state. One

[4]Deutsch's original paper [Deu89] shows that repeated applications of single-qubit rotations and the CNOT operation (described in §16.3.2) are enough to approximate any unitary transformation.

particularly important rotation is the **Hadamard operator**

$$H = \sqrt{\frac{1}{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

This maps $|0\rangle$ to the superposition $\sqrt{\frac{1}{2}}|0\rangle + \sqrt{\frac{1}{2}}|1\rangle$; since this superposition collapses to either $|0\rangle$ or $|1\rangle$ with probability $1/2$, the state resulting from $H|0\rangle$ is the quantum-computing equivalent of a fair coin-flip. Note that $H|1\rangle = \sqrt{\frac{1}{2}}|0\rangle - \sqrt{\frac{1}{2}}|1\rangle \neq H|0\rangle$. Even though both yield the same probabilities, these two superpositions have different **phases** and may behave differently when operated on further. That $H|0\rangle$ and $H|1\rangle$ are different is necessary, and indeed a similar outcome occurs for any quantum operation: all quantum operations are **reversible**, because any unitary matrix U has an inverse U^*.

If we apply H in parallel to all the qubits in our system, we get the n-fold tensor product $H^{\otimes n}$, which (if we take our bit-vector indices as integers $0 \ldots N-1 = 2^n - 1$ represented in binary) maps $|0\rangle$ to $\sqrt{\frac{1}{N}} \sum_{i=0}^{N-1} |i\rangle$. So n applications of H effectively scramble a deterministic initial state into a uniform distribution across all states. We'll see this scrambling operation again when we look at Grover's algorithm in §16.5.

16.3.2 Quantum implementations of classical operations

One issue that comes up with trying to implement classical algorithms in the quantum-circuit model is that classical operation are generally not reversible: if I execution $x \leftarrow x \wedge y$, it may not be possible to reconstruct the old state of x. So I can't implement AND directly as a quantum operation.

The solution is to use more sophisticated reversible operations from which standard classical operations can be extracted as a special case. A simple example is the **controlled NOT** or **CNOT** operator, which computes the mapping $(x, y) \mapsto (x, x \oplus y)$. This corresponds to the matrix (over the basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

which is clearly unitary (the rows are just the standard basis vectors). We could also write this more compactly as $|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|$.

The CNOT operator gives us XOR, but for more destructive operations we need to use more qubits, possibly including junk qubits that we won't look at again but that are necessary to preserve reversibility. The **Toffoli gate** or **controlled controlled NOT** gate (**CCNOT**) is a 3-qubit gate that was originally designed to show that classical computation could be performed reversibly [Tof80]. It implements the mapping $(x, y, z) \mapsto (x, y, (x \wedge y) \oplus z)$, which corresponds to the 8×8 matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

By throwing in some extra qubits we don't care about, Toffoli gates can implement basic operations like NAND ($(x, y, 1) \mapsto (x, y, \neg(x \wedge y))$), NOT ($(x, 1, 1) \mapsto (x, 1, \neg x)$), and fan-out ($(x, 1, 0) \mapsto (x, 1, x)$).[5] This gives a sufficient basis for implementing all classical circuits.

What we get at the end of applying this process to a classical circuit implementing some Boolean function f is a quantum circuit implementing a unitary operator f that maps $|x, y\rangle$ to $|x, y \oplus f(x)\rangle$ (possibly with some additional auxiliary qubits getting passed through as well). If we want to treat this operator as an input to a later computation (say, one that tests some condition on all of f), there is a further simplification step that we can do that will make f much easier to work with. This replaces the XOR $y \oplus f(x)$ with a change in phase on $|x\rangle$ itself.

[5] In the case of fan-out, this only works with perfect accuracy for classical bits and not superpositions, which run into something called the **no-cloning theorem**. For example, applying CCNOT to $\frac{1}{\sqrt{2}}|010\rangle + \frac{1}{\sqrt{2}}|110\rangle$ yields $\frac{1}{\sqrt{2}}|010\rangle + \frac{1}{\sqrt{2}}|111\rangle$. This works, sort of, but the problem is that the first and last bits are still entangled, meaning we can't operate on them independently. This is actually not all that different from what happens in the probabilistic case (if I make a copy of a random variable X, it's correlated with the original X), but it has good or bad consequences depending on whether you want to prevent people from stealing your information undetected or run two copies of a quantum circuit on independent replicas of a single superposition.

16.3.3 Phase representation of a function

The idea of a phase representation of a Boolean function f is to use an operator $U_f = \sum_x (-1)^{f(x)} |x\rangle \langle x|$. This has the effect of mapping each $|x\rangle$ to $-|x\rangle$ when $f(x) = 1$, and passing it through intact when $f(x) = 0$. The result is a diagonal matrix whose diagonal looks like a truth table for f expressed as ± 1 values. This has the effect of mapping each $|x\rangle$ to $-|x\rangle$ when $f(x) = 1$, and passing it through intact when $f(x) = 0$. The result is a diagonal matrix U_f whose diagonal looks like a truth table for f expressed as ± 1 values, as in this matrix for the XOR function $f(x_0, x_1) = x_0 \oplus x_1$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Because the amplitude of each $|x\rangle$ is unchanged, this representation is not so useful for observing the value of $f(x)$ directly. So we will generally be using this representation of f in the context of some larger quantum algorithm that exploits the phase changes to get useful cancellations for some values of x.

The question remains how we get the phase representation U_f, given that our standard construction of a classical circuit gives the controlled-not representation $f : |xy\rangle \mapsto |x, y \oplus f(x)\rangle$. The trick here is to feed this f the particular value $y = H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ This gives

$$\begin{aligned} f|xy\rangle &= \frac{1}{\sqrt{2}}(f|x0\rangle - f|x1\rangle) \\ &= \frac{1}{\sqrt{2}}(|x, f(x)\rangle - |x, \neg f(x)\rangle) \\ &= \frac{(-1)^{f(x)}}{\sqrt{2}}(|x0\rangle - |x1\rangle). \\ &= (-1)^{f(x)}|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \\ &= (U_f|x\rangle)|y\rangle. \end{aligned}$$

By leaving the auxiliary y alone in subsequent computation, we effectively end up with just $U_f|x\rangle$.

16.3.4 Practical issues (which we will ignore)

The big practical question is whether any of these operations—or even non-trivial numbers of independently manipulable qubits—can be implemented in a real, physical system. As theorists, we can ignore these issues, but in real life they are what would make quantum computing science instead of science fiction.[6]

16.3.5 Quantum computations

Putting everything together, a quantum computation consists of three stages:

1. We start with a collection of qubits in some known state x_0 (e.g., $|000\ldots0\rangle$).
2. We apply a sequence of unitary operators $A_1, A_2, \ldots A_m$ to our qubits. Some of these unitary operators may be oracles U_f or U_w representing the input we actually care about.
3. We take a measurement of the final superposition $A_m A_{m-1} \ldots A_1 x_0$ that collapses it into a single state, with probability equal to the square of the amplitude of that state.

Our goal is for this final state to tell us what we want to know, with reasonably high probability.

16.4 Deutsch's algorithm

We now have enough machinery to describe a real quantum algorithm. Known as Deutsch's algorithm, this computes $f(0) \oplus f(1)$ while evaluating f once [Deu89]. The trick, of course, is that f is applied to a superposition.

Assumption: f is implemented as a unitary operator U_f that that maps $|x\rangle$ to $(-1)^{f(x)}|x\rangle$. To compute $f(0) \oplus f(1)$, evaluate

$$\begin{aligned} HU_fH|0\rangle &= \sqrt{\frac{1}{2}} HU_f(|0\rangle + |1\rangle) \\ &= \sqrt{\frac{1}{2}} H\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right) \\ &= \frac{1}{2}\left(\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle\right). \end{aligned}$$

[6] Current technology, sadly, still puts quantum computing mostly in the science fiction category.

Suppose now that $f(0) = f(1) = b$. Then the $|1\rangle$ terms cancel out and we are left with

$$\frac{1}{2}\left(2 \cdot (-1)^b |0\rangle\right) = (-1)^b |0\rangle .$$

This puts all the weight on $|0\rangle$, so when we take our measurement at the end, we'll see 0.

Alternatively, if $f(0) = b \neq f(1)$, it's the $|0\rangle$ terms that cancel out, leaving $(-1)^b |1\rangle$. The phase depends on b, but we don't care about the phase. The important thing is that if we measure the qubit, we always see 1.

The result in either case is that with probability 1, we determine the value of $f(0) \oplus f(1)$, after evaluating f once (albeit on a superposition of quantum states).

This is kind of a silly example, because the huge costs involved in building our quantum computer almost certainly swamp the factor-of-2 improvement we got in the number of calls to f. But a generalization of this trick, known as the Deutsch-Josza algorithm [DJ92], solves the much harder (although still a bit contrived-looking) problem of distinguishing a constant Boolean function on n bits from a function that outputs one for exactly half of its inputs. No deterministic algorithm can solve this problem without computing at least $2^n/2 + 1$ values of f, giving an exponential speed-up.

The speed-up compared to a randomized algorithm that works with probability $1 - \epsilon$ is less impressive. With randomization, we only need to look at $O(\log 1/\epsilon)$ values of f to see both a 0 and a 1 in the non-constant case. But even here, the Deutsch-Josza algorithm does have the advantage of giving the correct answer with probability 1. If we make the same demand of a randomized algorithm, it does no better than a deterministic algorithm, at least in the constant-function case.

16.5 Grover's algorithm

Grover's algorithm [Gro96] is one of two main exemplars for the astonishing power of quantum computers.[7] The idea of Grover's algorithm is that if we have a function f on $N = 2^n$ possible inputs whose value is 1 for exactly one possible input w, we can find this w with high probability using $O(\sqrt{N})$ quantum evaluations of f. As with Deutsch's algorithm, we assume that f

[7] The other is Shor's algorithm [Sho97], which allows a quantum computer to factor n-bit integers in time polynomial in n. Sadly, Shor's algorithm is a bit too complicated to talk about here.

is encoded as an operator (conventionally written U_w) that maps each $|x\rangle$ to $(-1)^{f(x)}|x\rangle$.

The basic outline of the algorithm:

1. Start in the superposition $|s\rangle = \sqrt{\frac{1}{N}}\sum_x |x\rangle = H^{\otimes n}|0\rangle$.
2. Alternate between applying the **Grover diffusion operator** $D = 2|s\rangle\langle s| - I$ and the f operator $U_w = 2|w\rangle\langle w| - I$. Do this $O(\sqrt{n})$ times (the exact number of iterations is important and will be calculated below).
3. Take a measurement of the state. It will be w with high probability.

Making this work requires showing that (a) we can generate the original superposition $|s\rangle$, (b) we can implement D efficiently using unitary operations on a constant number of qubits each, and (c) we actually get w at the end of this process.

16.5.1 Initial superposition

To get the initial superposition, start with $|0^n\rangle$ and apply the Hadamard transform to each bit individually; this gives $\sqrt{\frac{1}{N}}\sum_x |x\rangle$ as claimed.

16.5.2 The Grover diffusion operator

We have the definition $D = 2|s\rangle\langle s| - I$.

Before we try to implement this, let's start by checking that it is in fact unitary. Compute

$$\begin{aligned} DD^* &= (2|s\rangle\langle s| - I)^2 \\ &= 4|s\rangle\langle s|s\rangle\langle s| - 4|s\rangle\langle s| + I^2 \\ &= 4|s\rangle\langle s| - 4|s\rangle\langle s| + I \\ &= I. \end{aligned}$$

Here we use the fact that $|s\rangle\langle s|s\rangle\langle s| = |s\rangle\langle s|s\rangle\langle s| = |s\rangle(1)\langle s| = |s\rangle\langle s|$.

Now let's look at implementation. Recall that $|s\rangle = H^{\otimes n}|0^n\rangle$, where $H^{\otimes n}$ is the result of applying H to each of the n bits individually. We also have that $H^* = H$ and $HH^* = I$, from which $H^{\otimes n}H^{\otimes n} = I$ as well.

So we can expand

$$\begin{aligned} D &= 2\left|s\right\rangle\left\langle s\right| - I \\ &= 2H^{\otimes n}\left|0^n\right\rangle\left(H^{\otimes n}\left|0^n\right\rangle\right)^* - I \\ &= 2H^{\otimes n}\left|0^n\right\rangle\left\langle 0^n\right| H^{\otimes n} - I \\ &= H^{\otimes n}\left(2\left|0^n\right\rangle\left\langle 0^n\right| - I\right) H^{\otimes n}. \end{aligned}$$

The two copies of $H^{\otimes n}$ involve applying H to each of the n bits, which we can do. The operator in the middle, $2\left|0^n\right\rangle\left\langle 0^n\right| - I$, maps $\left|0^n\right\rangle$ to $\left|0^n\right\rangle$ and maps all other basis vectors $\left|x\right\rangle$ to $-\left|x\right\rangle$. This can be implemented as a NOR of all the qubits, which we can do using our tools for classical computations. So the entire operator D can be implemented using $O(n)$ qubit operations, most of which can be done in parallel.

16.5.3 Effect of the iteration

To see what happens when we apply $U_w D$, it helps to represent the state in terms of a particular two-dimensional basis. The idea here is that the initial state $\left|s\right\rangle$ and the operation $U_w D$ are symmetric with respect to any basis vectors $\left|x\right\rangle$ that aren't $\left|w\right\rangle$, so instead of tracking all of these non-$\left|w\right\rangle$ vectors separately, we will represent all of them by a single composite vector

$$\left|u\right\rangle = \sqrt{\frac{1}{N-1}} \sum_{x \neq w} \left|x\right\rangle.$$

The coefficient $\sqrt{\frac{1}{N-1}}$ is chosen to make $\left\langle u|u\right\rangle = 1$. As always, we like our vectors to have length 1.

Using $\left|u\right\rangle$, we can represent

$$\left|s\right\rangle = \sqrt{\frac{1}{N}}\left|w\right\rangle + \sqrt{\frac{N-1}{N}}\left|u\right\rangle. \tag{16.5.1}$$

A straightforward calculation shows that this indeed puts $\sqrt{\frac{1}{N}}$ amplitude on each $\left|x\right\rangle$.

Now we're going to bring in some trigonometry. Let $\theta = \sin^{-1}\sqrt{\frac{1}{N}}$, so that $\sin\theta = \sqrt{\frac{1}{N}}$ and $\cos\theta = \sqrt{1-\sin^2\theta} = \sqrt{\frac{N-1}{N}}$. We can then rewrite (16.5.1) as

$$\left|s\right\rangle = (\sin\theta)\left|w\right\rangle + (\cos\theta)\left|u\right\rangle. \tag{16.5.2}$$

Let's look at what happens if we expand D using (16.5.2):

$$\begin{aligned}
D &= 2\left|s\right\rangle\left\langle s\right| - I \\
&= 2\left((\sin\theta)\left|w\right\rangle + (\cos\theta)\left|u\right\rangle\right)\left((\sin\theta)\left\langle w\right| + (\cos\theta)\left\langle u\right|\right) - I \\
&= (2\sin^2\theta - 1)\left|w\right\rangle\left\langle w\right| + (2\sin\theta\cos\theta)\left|w\right\rangle\left\langle u\right| + (2\sin\theta\cos\theta)\left|u\right\rangle\left\langle w\right| + (2\cos^2\theta - 1)\left|u\right\rangle\left\langle u\right| \\
&= (-\cos 2\theta)\left|w\right\rangle\left\langle w\right| + (\sin 2\theta)\left|w\right\rangle\left\langle u\right| + (\sin 2\theta)\left|u\right\rangle\left\langle w\right| + (\cos 2\theta)\left|u\right\rangle\left\langle u\right| \\
&= \begin{bmatrix} -\cos 2\theta & \sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{bmatrix},
\end{aligned}$$

where the matrix is over the basis $(\left|w\right\rangle, \left|u\right\rangle)$.

Multiplying by U_w negates all the $\left|w\right\rangle$ coefficients. So we get

$$\begin{aligned}
U_w D &= \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} -\cos 2\theta & \sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{bmatrix} \\
&= \begin{bmatrix} \cos 2\theta & -\sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{bmatrix}. \qquad (16.5.3)
\end{aligned}$$

Aficionados of computer graphics, robotics, or just matrix algebra in general may recognize (16.5.3) as the matrix that rotates two-dimensional vectors by 2θ. Since we started with $\left|s\right\rangle$ at an angle of θ, after t applications of this matrix we will be at an angle of $(2t+1)\theta$, or in state

$$(\sin(2t+1)\theta)\left|w\right\rangle + (\cos(2t+1)\theta)\left|u\right\rangle.$$

Ideally, we pick t so that $(2t+1)\theta = \pi/2$, which would put all of the amplitude on $\left|w\right\rangle$. Because t is an integer, we can't do this exactly, but setting $t = \lfloor\frac{\pi/2\theta - 1}{2}\rfloor$ will get us somewhere between $\pi/2 - 2\theta$ and $\pi/2$. Since $\theta \approx \sqrt{\frac{1}{N}}$, this gives us a probability of seeing $\left|w\right\rangle$ in our final measurement of $1 - O(\sqrt{1/N})$ after $O(\sqrt{N})$ iterations of $U_w D$.

Sadly, this is as good as it gets. A lower bound of Bennet *et al.* [BBBV97] shows that *any* quantum algorithm using U_w as the representation for f must apply U_w at least $\Omega(\sqrt{N})$ times to find w. So we get a quadratic speedup but not the exponential speedup we'd need to solve **NP**-complete problems directly.

Chapter 17

Randomized distributed algorithms

A **distributed algorithm** is one that runs on multiple machines that communicate with each other in some way, typically via **message-passing** (an abstraction of packet-based computer networks) or **shared memory** (an abstraction of multi-core CPUs and systems with a common memory bus). What generally distinguishes **distributed computing** from the closely-related idea of **parallel computing** is that we expect a lot of nondeterminism in a distributed algorithm: events take place at unpredictable times, processes may crash, and in particular bad cases we may have **Byzantine processes** that work deliberately against the algorithm. We are not going to assume Byzantine processes; instead, we'll look at a particular model called **wait-free shared memory**.

This hostile nondeterminism is modeled by an **adversary** that controls scheduling and failures. For a shared-memory model, we have a collection of processes that each have a **pending operation** that is either a read or write to some **register**. The adversary chooses which of these pending operations happens next (so that concurrency between processes is modeled by interleaving their operations). Unlike the adversary that supplies the worst-case input to a traditional algorithm before it executes, an adversary scheduler might be able to observe the execution of a distributed algorithm in progress and adjust its choices in response. If it has full knowledge of the system (including internal states of processes), we call it an **adaptive adversary**; at the other extreme is an **oblivious adversary** that chooses the schedule of which processes execute operations at which times in advance. This is similar to the distinction between an adaptive and oblivious adversary

required for the analysis of randomized data structures (see §6.3.1.

For either adversary, what makes the system **wait-free** is that any process that gets to take an unbounded number of steps has to finish whatever it is doing no matter how the other processes are scheduled. In particular, this means that no process can wait for any of the others, because they might never be scheduled. The processes that do run all run the algorithm correctly, so the model is equivalent to assuming up to $n - 1$ of the processes may **crash** but that none are Byzantine.

As with traditional algorithms, distributed algorithms can use randomization to make themselves less predictable. The assumption is that even if the adversary is adaptive, it can't predict the outcome of future coin-flips. For some problems, avoiding such predictability is provably necessary.

Distributed computing is a huge field, and we aren't going to be able to cover much of it in the limited space we have here. So we are going to concentrate on a few simple problems that give some of the flavor or randomized distributed algorithms (and that lead to problems that are still open).

17.1 Consensus

The **consensus** problem is to get a collection of n processes to agree on a value. The requirements are that all the processes that don't crash finish the protocol (with probability 1 for randomized protocols) (**termination**), that they all output the same value (**agreement**), and that this value was an input to one of the processes (**validity**)—this last condition excludes protocols that always output the same value no matter what happens during the execution, and makes consensus useful for choosing among values generated by some other process.

There are many versions of consensus. The original problem as proposed by Pease, Shostak, and Lamport [PSL80] assumes Byzantine processes in a synchronous message-passing system. Here scheduling is entirely predictable, and the obstacle to agreement is dissension sown by the lying Byzantine processes. We will instead consider wait-free shared-memory consensus, where scheduling is unpredictable but the the processes and memory are trustworthy. Even in this case, the unpredictable scheduling makes solving the problem deterministically impossible.

17.1.1 Impossibility of deterministic algorithms

The FLP impossibility result [FLP85] and its extension to shared memory [LAA87] show that consensus is impossible to solve in an asynchronus model with even one crash failure. The general proof is a bit involved, but there is a simple intuitive proof of this result for the wait-free case where up to $n-1$ of the processes may crash.

Crash all but two processes, one with input 0 and one with input 1. Define the **preference** of a process as the value it will decided in a solo execution (this is well-defined because the processes are deterministic). In the initial state, the process with input b has preference b because of validity—it doesn't know the other process exists. But before it returns b, it has to cause the other process to change its preference (once it leaves the other process is running alone). It can do so only by sending a message or writing a register. When it is about to do this, stop it and run the other process until it does the same thing.

Either the other process somehow neutralizes the effect of the delayed message/write during this time, or it doesn't. In the first case, restart the first process (which still has its original preference and now must do something else to make the other process change). In the second case, deliver both operations and let the processes exchange preferences. The resulting execution looks very much like when two people are trying to pass each other in a hallway and oscillate back-and-forth—but since our process's have their timing controlled by an adversary, the natural damping or randomness that occurs in humans doesn't ever resolve the situation.

We can avoid this impossibility result with randomness. If either of the process decides to choose a new preference at random, then there is a 50% chance it will end up agreeing with the other one, no matter what the other one is doing. Assuming we can detect this agreement, this solves consensus in constant expected time for two processes.

Unfortunately, this flip-if-confused approach does not generalize very well to n processes. The first known randomized shared-memory algorithm for consensus, due to Abrahamson [Abr88], used essentially this idea. But because it required waiting for a long run of identical coin-flips across all processes, it took $O\left(2^{n^2}\right)$ steps on average to finish. In a little while, we will see how to do better.

17.2 Leader election

In a shared-memory system, **leader election** is a very similar problem to consensus. Here we want one of the processes to decide that it is the leader, and the rest to decide that they are followers. The difference from consensus is that we don't require that the followers learn the identity of the leader. This means that followers can drop out before the algorithm determines the leader, which may allow a faster algorithm.

Note however that if there are only two processes, the losing process knows who the winner is, because there is only one possibility. So the same argument that shows that we can't do consensus deterministically with two processes also works for leader election.

17.3 How randomness helps

We've already hinted at the idea of using randomness to shake processes into agreement. In the current literature on shared-memory consensus and leader election, this idea shows up in two main forms:

1. We can replace the n separate coin-flips of stalled processes with a single **weak shared coin** protocol. This is a protocol that has for each outcome $b \in \{0, 1\}$ a minimum probability $\delta > 0$ that every processes receives b as the value of the coin. Because δ will typically be less than $1/2$, it is possible that we get disagreements, or that the adversary can bias the coin toward a particular value that it likes. But with a bit of extra machinery that we will not describe here, we can get agreement after $O(1/\delta)$ calls to the shared coin on average [AH90]. An example of a weak shared coin protocol that assumes an adaptive adversary is given in §17.4.

2. We can eliminate processes quickly using a **sifter**, which solves a weak version of leader election that allows for multiple winners but guarantees that the expected number of winners is small relative to n. Repeated application of a sifter can quickly knock the number of repeated winners down to 1, which solves leader election (§17.5. With some tinkering, it is possible to use some sifters to also solve consensus (§17.6. In both cases we assume an oblivious adversary.

17.4 Building a weak shared coin

The goal of a weak shared coin is to minimize the influence of the adversary scheduler. An adaptive adversary can bias the outcome of the coin by looking at the random values generated by the individual processes, and withholding values that it doesn't like by delaying any write operations by those processes. So we would like to find a way to combine the local coins at the processes together into a single global coin that minimizes the influence of any particular local coin. The easiest way to do this is by applying the majority function: each process repeatedly generates ± 1 values with equal probability and adds them to a common pile. When the pile includes enough local coins, we take the sign of the total to get the return value of the global coin.

The nice thing about this approach is that the behavior of the processes is symmetric. It doesn't matter what order the adversary runs them in, the next coin-flip is always another fair coin-flip. So we can analyze the sequence of partial sums of *generated* coin-flips using the tools we have developed for sums of independent variables, random walks, and martingales. Unfortunately things get a little more complicated when we look at the totals actually observed by any particular processes.

The problem is that we need some mechanism for gathering up the local coin values. Using read-write registers, we can have each process write the count and sum of its own local coins to a register writable by it alone (this avoids processes overwriting each other). Because the adversary can only delay one coin from each process, both total count and the total sum are always within n of the correct value. If we could read all the registers instantaneously and stop as soon as the count reached some threshold T, then for $T = \Theta(n^2)$ the distribution on the generated coins would be spread out enough that the total would be at least $n + 1$ away from 0 with constant probability. But the actual situation is more complicated.

The problem is that each process must do n sequential reads to read all the registers. This is not only expensive—it's not something we want to do after every local flip—but it also means that more coins may be generated while I'm reading the registers, meaning any sum I compute at the end of the algorithm might have only a tenuous connection with the actual sum of the generated coins at any point in the execution.

The solution to the first part of this problem was proposed by Bracha and Rachman [BR91]: only check the total after writing out $\Theta(n/\log n)$ local coin-flips, giving an amortized cost per coin-flip of $\Theta(\log n)$ register operations. Unfortunately this makes the second problem worse: a process p

might continue generating many local coins after the total count crosses the threshold, and each other process might see a different prefix of these coins depending on when it reads p's register.

Bracha and Rachman showed that this wasn't too much of a problem using Azuma's inequality (this is where the $O(\log n)$ factor comes in). But later work by Attiya and Censor [AC08] allowed for a simpler analysis of a slightly different algorithm, which we will describe here.

Pseudocode for the Attiya-Censor coin is given in Algorithm 17.1. The algorithm only checks the total count once for every n coin-flips, giving an amortized cost of 1 read per coin-flip. But to keep the processes for running too long, each process checks a multi-writer stop bit after ever coin-flip.

```
1 while ∑_{j=1}^{n} r_j.count < T do
2     for i ← 1 . . . n do
3         ⟨r_i.count, r_i.sum⟩ ← ⟨r_i.count + 1, r_i.sum ± 1⟩
4         if stop then
5             break;
6     stop ← true
7     return sgn(∑_{j=1}^{n} r_j.sum)
```

Algorithm 17.1: Attiya-Censor weak shared coin [AC08]

Each process may see a different total sum at the end, but our hope is that if T is large enough, there is at least a δ probability that all these total sums are positive (or, by symmetry, negative). We can represent the total sum seen by any particular process i as a sum $S + D + X_i - H_i$, where:

1. S is the sum of the first T coin-flips. This has a binomial distribution, equal to the sum of T independent ± 1 random variables. Letting $T = cn^2$ for some reasonably large c gives us a constant probability that $|S| > 4n$.

2. D is the sum of all coin-flips after the first T that are generated before some process sets the stop bit. There are at most $n^2 + n$ such coin-flips, and they form a martingale with bounded increments. So Azuma's inequality gives us that $|D| \leq 2n$ with at least constant probability, independent of S.

3. $X_i = \sum_{j=1}^{n} Y_{ij}$, where Y_{ij} is the sum of all coin-flips generated by process j between some process setting the stop bit and process i

reading r_j. Since each process can generate at most one extra coin-flip before checking stop, $|X_i| \leq n$ always.

4. $H_i = \sum_{j=1}^{n} Z_{ij}$, where Z_{ij} is the sum of all votes that are generated by process j before i reads r_j, but that are not included in r_j.sum because they haven't been written yet. Again, each process can contribute only one coin-flip to H_i. So $|H_i| \leq n$ always.

Adding up the error terms $D + X_i - H_i$ gives a total that is bounded by $4n$ with at least constant probability. This probability is independent of the constant probability that $|S| > 4n$. So if both of these events occur, we win.

The total cost of the coin is $O(T + n^2) = O(n^2)$. This also gives a cost of $O(n^2)$ for consensus.

This turns out to be optimal, because of a lower bound on the expected total steps for consensus appearing in the same paper [AC08]. This is a bit disappointing, because an $\Omega(n^2)$ lower bound translates to $\Omega(n)$ steps for each process even if we can distribute the load evenly across all processes (which Algorithm 17.1 might not). We'd like to get something that scales better, but to get around the lower bound we will need to switch to an oblivious adversary.

17.5 Leader election with sifters

The idea of a **sifter** [AA11] is to use randomization to quickly knock out processes while leaving at least one process always. The current best known sifter for standard read-write registers, due to Giakkoupis and Woelfel [GW12], is shown in Algorithm 17.2.

```
1 Choose r ∈ ℤ⁺ such that Pr[r = i] = 2^{-i}
2 A[r] ← 1
3 if A[r + 1] = 0 then
4 |   stay
5 else
6 |   leave
```

Algorithm 17.2: Giakkoupis-Woelfel sifter [GW12]

The cost of executing the sifter is two operations. Each process chooses an index r according to a geometric distribution, writes $A[r]$, and then checks if any other process has written $A[r+1]$. The process stays if it sees nothing.

Because any process with the maximum value of r always stays, at least one process stays. To show that not too many processes stay, let X_i be the number of survivors with $r = i$. This will be bounded by the number of processes that write to $A[i]$ before any process writes to $A[i+1]$. We can immediately see that $\mathrm{E}[X_i] \le n \cdot 2^{-i}$, since each process has probability 2^{-i} of writing to $A[i]$. But we can also argue that $\mathrm{E}[X_i] \le 2$ for any value of n.

The reason is that because the adversary is oblivious, the choice of which process writes next is independent of the location it writes to. If we condition on some particular write being to either $A[i]$ or $A[i+1]$, there is a $1/3$ chance that it writes to $A[i+1]$. So we can think of the subsequence of writes that either land in $A[i+1]$ or $A[i]$ as a sequence of biased coin-flips, and we are counting how many probability-$2/3$ tails we get before the first probability-$1/3$ heads. This will be 2 on average, or at most 2 if we take into account that we will stop after n writes.

We thus have $\mathrm{E}[X_i] \le \min(2, n \cdot 2^{-i})$. So the expected total number of winners is bounded by $\sum_{i=1}^{\infty} \mathrm{E}[X_i] \min(2, n \cdot 2^{-i}) = 2 \lg n + O(1)$.

Now comes the fun part. Take all the survivors of our sifter, and run them through more sifters. Let S_i be the number of survivors of the first i sifters. We've shown that $\mathrm{E}[S_{i+1} \mid S_i] = O(\log S_i)$. Since log is concave, Jensen's inequality then gives $\mathrm{E}[S_{i+1}] = O(\log \mathrm{E}[S_i])$. Iterating this gives $\mathrm{E}[S_i] = O(\log^{(i)} S_0) = O(\log^{(i)} n)$. So there is some $i = O(\log^* n$ at which $\mathrm{E}[S_i]$ is a constant.

This doesn't quite give us leader election because the constant might not be 1. But there are known leader election algorithms that run in $O(1)$ time with $O(1)$ expected participants [AAG+10], so we can use these to clean up any excess processes that make it through all the sifters. The total cost is $O(\log^* n)$ operations for each process.

17.6 Consensus with sifters

We've seen that we can speed up leader election by discarding potential leaders quickly. For consensus, we want to discard potential output values quickly. This is a little more complicated, because processes carrying losing output values can't just drop out if we haven't determined the winning values yet.

We are going to look at an algorithm by Aspnes and Er [AE19] that solves consensus in $O(\log^* n)$ operations per process, using sifters built on top of a stronger primitive than normal registers. (This algorithm uses some ideas from an earlier paper of Aspnes [Asp15], which gives a more

complicated $O(\log \log n)$-time algorithm for standard registers.) As with the Giakkoupis-Woelfel leader election algorithm, we assume an oblivious adversary scheduler.

The stronger primitive we need is a **max register** [AACH12]. Unlike a standard register, reading a max register returns the largest value ever written to it; equivalently, trying to write a smaller value to a max register than it already contains has no effect.

What this means for randomized algorithms is that we can knock out values using a max register by a simple priority scheme. If each process writes a tuple $\langle r, v \rangle$ to the max register, where r is a unique random priority and v is the process's preferred value, then the i-th value written appears in the max register only if it is a left-to-right maximum of the sequence of priorities, which occurs with probability roughly $1/i$. (This follows from symmetry and the fact that the oblivious adversary can't selectively schedule smaller priorities first.) So on average only $H_n = O(\log n)$ of these values will ever appear in the max register. By having each process read and return a value from the max register, we've gone from n possible values to $O(\log n)$ possible values on average.

We now run into two technical obstacles. The first is that we can't necessarily guarantee that all the random priorities are unique, which we would need to get an exact $1/i$ probability for survival. This can be dealt with by choosing priorities over a sufficiently large range that the rare collisions don't add much to the expected survivors (say, $O(n^3)$). The second issue is that if we try to iterate the sifter, in the second round we have many copies of each input value. So even if one copy drops out because of a low priority, some other copy might get lucky and survive. To go from $O(\log n)$ expected values to $O(\log \log n)$ expected values, we need to make sure that no value gets more than one chance at surviving.

Here is where we use an idea that first appeared in [Asp15]. Instead of generating a new priority for each value in each round, we generate a sequence $r_1, r_2, \ldots, r_\ell$ of priorities for all ℓ rounds all at once at the beginning. We can then store $\langle r_i, \ldots, r_\ell, v \rangle$ in the max register for round i, and the leading priority r_i controls survival. Now we don't care about having multiple copies of a value, because they will all have the same initial r_i, and if the first copy to be written doesn't survive, none of them will.

The result of applying this idea is Algorithm 17.3. Because each iteration of the loop reduces S_i survivors to $S_{i+1} = O(\log S_i)$ survivors on average, the same application of Jensen's inequality that we used for Algorithm 17.2 works here as well, so for an appropriate $\ell = O(\log^* n)$ we can get down to a single surviving value with reasonably high probability.

```
1 procedure sifter(v)
2     Choose random ranks r_1 ... r_ℓ
3     for i ← 1 ... ℓ do
4         writeMax(M_i, ⟨r_i, ..., r_ℓ, v⟩)
5         ⟨r_i, ..., r_ℓ, v⟩ ← readMax(M_i)
6     return v
```

Algorithm 17.3: Sifter using max registers [AE19]

This is not quite enough to get consensus, which requires agreement always, but with some additional machinery it is possible to detect when the protocol fails and re-run it if necessary. The final result is consensus in expected $O(\log^* n)$ max register operations.

Appendix A

Sample assignments from Spring 2024

A.1 Assignment 1, due Thursday 2024-02-01 at 23:59

A.1.1 Matchings

A **matching** in a graph is a subgraph where each vertex has degree at most 1, which means that some vertices are paired with other vertices. In this problem, we will consider several randomized algorithms for quickly constructing a large matching in a d-regular graph. For each algorithm, your task is to compute an exact closed-form expression for the expected number of edges $M(n, d)$ in the matching as a function of n and d.

1. In the simplest algorithm, each vertex u chooses one of its d neighbors v independently and uniformly at random. Each edge uv is included in the matching if u chooses v and v chooses u.

2. The second algorithm assumes a bipartite d-regular graph where n is even and the nodes are partitioned into two subsets S and T of $n/2$ nodes each, with all edges going between S and T. In this graph, have each node u in S choose one of its d neighbors v in T independently and uniformly at random. Each edge uv is included in the matching if u chooses v and no other u' chooses v.

3. The third algorithm attempts to extend the bipartite-graph algorithm to a general graph. Each node u flips an independent fair coin to

decide whether it will send or receive. Each sender then picks one of its d neighbors independently and uniformly at random. An edge uv is included in the matching if u is a sender, v is a receiver, and u is the only sender that picks v; or if the same conditions hold with u and v reversed.

Solution

In each case we'll use linearity of expectation.

1. Let X_{uv} be the indicator for the event that uv is included in the matching. Then

$$\begin{aligned}\mathrm{E}[X_{uv}] &= \Pr[u \text{ and } v \text{ are matched}] \\ &= \Pr[u \text{ chooses } v \text{ and } v \text{ chooses } u] \\ &= \Pr[u \text{ chooses } v]\Pr[v \text{ chooses } u] \\ &= (1/d)(1/d) \\ &= 1/d^2.\end{aligned}$$

   Now sum over all $nd/2$ edges to get

$$M(n,d) = (nd/2)(1/d^2) = \frac{n}{2d}.$$

2. Again let X_{uv} be the indicator variable for the event that uv is included. Now

$$\begin{aligned}\mathrm{E}[X_{uv}] &= \Pr[\text{only } u \text{ picks } v] \\ &= (1/d)(1-1/d)^{d-1}.\end{aligned}$$

   Sum over all $nd/2$ edges to get

$$M(n,d) = (nd/2)(1/d)(1-1/d)^{d-1} = \frac{n}{2}(1-1/d)^{d-1}.$$

   Using $1 - 1/d \le e^{-1/d}$, we can show that this is at least $\frac{n}{2e}$ for any d, meaning we match a constant fraction of the nodes on average even when d is large.

3. For this version it's convenient to break symmetry and make X_{uv} be the indicator variable for the event that u is a sender, v is a receiver, and u is matched with v. This means that we will have two variables

X_{uv} and X_{vu} for each edge, but we can deal with this when we need to.

Compute

$$\begin{aligned}\mathrm{E}[X_{uv}] &= \Pr[u \text{ is sender and } v \text{ is receiver and only } u \text{ picks } v] \\ &= (1/2)(1/2)(1/d)\left(1 - \frac{1}{2d}\right)^{d-1} \\ &= \frac{1}{4d}\left(1 - \frac{1}{2d}\right)^{d-1}.\end{aligned}$$

Sum over all nd directed edges uv to get

$$M(n,d) = \frac{n}{4}\left(1 - \frac{1}{2d}\right)^{d-1}.$$

It can be shown that this is also $\Theta(n)$ regardless of d, although the constant is not quite as good as in the bipartite case.

A.1.2 Non-volatile memory

A manufacturer has designed a non-volatile memory chip based on blowing tiny fuses. Each fuse starts out representing a 0, but by running an excess voltage across the fuse it can be permanently changed to a new state representing a 1.

To allow storing multiple values in the chip, it is arranged as a sequence of cells of k bits each, with an extra bit used to mark the cell as discarded. Initially a cell will hold all zeros (0000). When writing a new value v, if it is possible to overwrite the previous value without having to turn any 1 into a 0, the same cell will be re-used. Otherwise it will be marked as discarded and a new cell will be used for v. An example of this process is given in Figure A.1.

In the worst case, every new value requires a new cell. But the manufacturer hopes that things will go better with enough randomness.

1. Suppose that a sequence $v_1, v_2, \ldots, v_n$ of $n \geq 1$ values is generated uniformly at random, so that each v_i is equally likely to be each of the 2^k possible bit-vectors of length k. Give an exact closed-form expression for the expected number of cells used to write these values, as a function of n and k.

Value	New memory contents
(Initial)	0000
0101	0101
1011	~~0101~~ 1011
0010	~~0101~~ ~~1011~~ 0010
0110	~~0101~~ ~~1011~~ 0110
1100	~~0101~~ ~~1011~~ ~~0110~~ 1100
1101	~~0101~~ ~~1011~~ ~~0110~~ 1101

Figure A.1: Non-volatile memory in action. The example shows $n = 6$ writes with $k = 4$ bits per cell. A total of 4 cells are used.

2. Since we can't rely on a random input, let's move the randomness into the algorithm. Suppose that a fixed permutation π on all k-bit vectors is chosen uniformly at random and each v_i (supplied by an adversary) is mapped to $\pi(v_i)$ before being written. Now what is the expected number of cells used in the worst case? (As usual, assume that π is chosen after $v_1, \ldots, v_n$ are fixed.)

3. The manufacturer complains that storing π is too expensive, and suggests storing a random k-bit vector x instead. Each v_i is now mapped to $v_i \oplus x$ before being stored, where $\oplus$ represents bitwise XOR. Now what is the expected number of cells used in the worst case?

Solution

1. Let v and v' be consecutive values. Then v' requires a new cell if there is some position j such that $v'_j = 0$ but $v_j = 1$. For each position j, this occurs with probability $1/4$ and does not occur with probability $3/4$. Because the positions are independent, the probability that v' does not require a new cell is $(3/4)^k$ and the probability that it does require a new cell is $1 - (3/4)^k$.

   For each $i > 1$, let X_i be the indicator variable for the even that v_i requires a new cell. If we let $X_1 = 1$, then $S = \sum_{i=1}^{n} X_i$ gives the total number of cells needed. We have calculated above that $\mathrm{E}[X_i] = 1 - (3/4)^k$ for $i > 1$, so by linearity of expectation

$$\begin{aligned}\mathrm{E}[S] &= \sum_{i=1}^{n} \mathrm{E}[X_i] \\ &= 1 + (n-1)(1 - (3/4)^k).\end{aligned}$$

2. Same setup as before but we need to recompute $\mathrm{E}[X_i]$. The problem is that $\pi(v)$ and $\pi(v')$ are no longer independent, because the adversary can (and should) pick $v \neq v'$, which removes one of the possible cases for $\pi(v')$ since π can't send unequal v and v' to the same value.

   We can compensate for this using conditional expectations. Let X be the indicator for the event that $\pi(v')$ requires a new cell if $\pi(v)$ and $\pi(v')$ are chosen independently at random. Then

   $$\begin{aligned} 1 - (3/4)^k &= \mathrm{E}[X] \\ &= \mathrm{E}[X \mid \pi(v) = \pi(v')] \Pr[\pi(v) = \pi(v')] \\ &\quad + \mathrm{E}[X \mid \pi(v) \neq \pi(v')] \Pr[\pi(v) \neq \pi(v')] \\ &= 0 \cdot 2^{-k} + \mathrm{E}[X \mid \pi(v) \neq \pi(v')] (1 - 2^{-k}), \end{aligned}$$

   which we can solve to get

   $$\mathrm{E}[X \mid \pi(v) \neq \pi(v')] = \frac{1 - (3/4)^k}{1 - 2^{-k}}.$$

   Since the only choice the adversary can make that affects $\mathrm{E}[X_i]$ or not is to set $v_i \neq v_{i-1}$, this gives a worst case of

   $$\mathrm{E}\left[\sum X_i\right] = 1 + (n-1)\frac{1 - (3/4)^k}{1 - 2^{-k}},$$

   which is only a little bit worse than the average case.

   An alternative approach to computing $\mathrm{E}[X]$ is to just count the number of pairs $\pi(v), \pi(v')$ that require a new cell and divide by the total number of possibilities $2^k(2^k - 1)$. Here there are 3 choices for each bit position that don't require a new cell, giving 3^k choices overall, but 2^k of these choices make $\pi(v) = \pi(v')$. So the number of distinct pairs that don't require a new cell is $3^k - 2^k$. We can get the number of pairs that do require a new cell by subtracting from the total $2^k(2^k - 1) = 4^k - 2^k$. This gives $\mathrm{E}[X] = \frac{4^k - 3^k}{4^k - 2^k} = \frac{1-(3/4)^k}{1-2^{-k}}$ as computed above.

3. Here things get substantially worse. Suppose that $v = 0^k$ and $v' = 1^k$. Now $v' \oplus x$ requires a new cell as long as $x \neq 0^k$, which occurs with probability $1 - 2^{-k}$. A similar argument holds when $v = 1^k$ and $v' = 0^k$. So by alternating between all-0 and all-1 values, the adversary can force us to use an expected $1 + (n-1)(1 - 2^{-k})$ cells. It can't do worse since any choice other than switching all bits would give a lower expected cost for each write.

All of these bounds are terrible, but they are not equally terrible and are modestly distinct for small k. For $k = 2$, for example, the coefficient on $(n-1)$ goes from $7/16$ in the average case, to $7/12$ in the permutation case, and all the way up to $3/4$ in the bitwise XOR case.

A.2 Assignment 2, due Thursday 2024-02-15 at 23:59

A.2.1 Mediocre cuts

Recall that a cut in a graph G is a partition of the vertices into sets S and T, where the size of the cut is the number of edges with one endpoint in each set.

Given a graph G with n vertices and m edges, we'd like to construct a cut with size as close as possible to $m/2$. We will consider two algorithms that generate 0-1 random variables X_v for each vertex v, such that v is in S if $X_v = 0$ and v is in T if $X_v = 1$.

1. In the first algorithm, X_v is simply an independent fair coin-flip for each v.

   Prove or disprove: This algorithm produces a cut of size $m/2 \pm o(m)$ with constant nonzero probability.

2. In this second algorithm, the independent fair coins are replaced by pairwise independent coins, constructed as in §5.1.2.1: Generate $k = \lceil \lg(n+1) \rceil$ independent fair random bits $Y_1, \ldots, Y_k$, assign each vertex v a unique non-empty subset A_v of these bits, and let $X_v = \bigoplus_{i \in A_v} Y_i$.

   Prove or disprove: This algorithm produces a cut of size $m/2 \pm o(m)$ with constant nonzero probability.

Solution

1. We'll give a proof using Chebyshev's inequality.

   For each edge uv let $Z_{uv} = X_u \oplus X_v$ be the indicator variable for the event that uv is in the cut. Let $S = \sum_{uv} Z_{uv}$ be the size of the cut.

   We have $\mathrm{E}[S] = \sum \mathrm{E}[Z_{uv}] = m/2$.

   To compute $\mathrm{Var}[S]$, we will first show that the variables Z_{uv} are pairwise-independent. This holds trivially for any variables corresponding to non-incident edges. For the case of two incident edges Z_{uv} and

Z_{vw}, observe that $\mathrm{E}[Z_{vw} \mid Z_{uv}] = 1/2$, because whatever value X_v has, adding X_w yields 0 or 1 with equal probability. So Z_{uv} and Z_{vw} are independent as well.

Since the Z_{uv} are pairwise-independent, $\mathrm{Var}[S] = \sum \mathrm{Var}[Z_{uv}] = m/4$. Chebyshev then says

$$\Pr[|S - m/2| \geq \sqrt{m}] \leq \frac{m/4}{(\sqrt{m})^2} = \frac{1}{4}.$$

This gives a 3/4 chance of getting S in the range $m/2 \pm \sqrt{m} = m/2 \pm o(m)$.

2. For this version we will give a disproof, by constructing a family of arbitrarily large graphs where the algorithm generates a cut of size 0 or m with equal probability.

   Fix some n, and create an edge uv for each pair of vertices u and v with $A_u = A_v \triangle \{1\}$. This will give a total of $m = n/2$ edges. Each such edge will have $X_u = X_v \oplus Y_1$; this will put uv in the cut precisely when $Y_1 = 1$. So the cut will have 0 edges with probability 1/2 and m edges with probability 1/2.

   In neither case is the size of the cut within $o(m)$ of $m/2$, so the algorithm does not guarantee this with nonzero probability.

A.2.2 Training costs

A total of n workers arrive at a site in blocks of size $n_1, n_2, \ldots, n_k$, where $\sum n_i = n$. After the i-th block arrives, one of the $n_1 + n_2 + \cdots + n_i$ workers now on site is chosen uniformly at random to be the leader. If the leader arrived previously (was one of the $n_1 + n_2 + \cdots + n_{i-1}$ workers already at the site), they are said to be *experienced* and require no training. If the leader was one of the most recent n_i arrivals, they are *inexperienced* and must be trained. We'd like to get an estimate of the worst-case expected cost of training inexperienced leaders, assuming that an adversary chooses the sizes and number of the blocks $n_1, \ldots, n_k$, and show that the actual cost is likely to be close to the expected cost.

We consider two different models for the cost of training an inexperienced leader.

1. In the first model, an inexperienced leader remedies their ignorance by calling the help line 1-800-LEADER, at a cost of one unit.

Show an upper bound $f(n)$ on the worst-case expected total training cost for this model, and show that there is are constants $c > 0$ and $c' < 1$ such that actual cost in this worst case is within $f(n) \pm c' f(n)$ for sufficiently large n with probability at least $1 - O(n^{-c})$.

2. **This part of the problem contained an error and has been withdrawn. You do not need to supply a solution to this part of the problem and will be given full credit for it.**

Solution

1. First we need to figure out the worst-case choice of $n_1, \ldots, n_k$.

   For any particular choice of the block sizes, the expected cost is

   $$\sum_{i=1}^{k} \frac{n_i}{\sum_{j=1}^{i} n_j}.$$

   It is not hard to see that this quantity is maximized when $k = n$ and $n_i = 1$ for all i. The proof is to observe that if any $n_i > 1$, it can be replaced by consecutive blocks of size $n_i - 1$ and 1, giving an expected cost for these two new blocks of

   $$\frac{n_i - 1}{\sum_{j=i} n_i - 1} + \frac{1}{\sum_{j=i}^{i} n_i} > \frac{n_i - 1}{\sum_{j=i} n_i} + \frac{1}{\sum_{j=i}^{i} n_i} = \frac{n_i}{\sum_{j=i} n_i},$$

   which increases the expected total cost.

   Setting each $n_i = 1$, the expected total cost is

   $$\sum_{i=1}^{n} \frac{1}{i} = H_n = \Theta(\log n).$$

   Let X_i be the training cost for the i-th block. Then each X_i is an independent Bernoulli random variable and $S = \sum X_i$ has known expectation $\mu = H_n = \Theta(\log n)$, so we can use the two-sided Chernoff bound (5.2.7) to get

   $$\Pr\left[|S - \mu| \geq (1/2)\mu\right] \leq 2e^{-\mu/12} = 2e^{-\Theta(\log n)} = O(n^{-c}),$$

   for some $c > 0$.

   But then S lies between $(1/2)\mu$ and $(3/2)\mu$ with probability $1 - O(n^{-c})$, giving the desired bound.

A.3 Assignment 3, due Thursday 2024-02-29 at 23:59

A.3.1 A robot rendezvous problem

A warehouse for a large online retailer has n packages, each of which has a distinct tracking number $t_i \in \{0, \ldots, t-1\}$. Each package starts out in the possession of one of n warehouse robots and needs to be transferred to one of n corresponding delivery drones to be delivered. We will refer to the robots and drones collectively as *workers*.

Unfortunately there are only $m < n$ landing pads for the drones, and the scheduling system that assigns packages to the workers does not know about the landing pads.

Instead, each worker knows only n, m, t, the unique tracking number t_i for its assigned package, whether it is a robot or a drone, and up to b bits of randomness that is shared between all the workers, where b is polylogarithmic in t (meaning that $b = O(\log^c t)$ for some c).[1] Using this information, it must choose one of the m landing pads $0, \ldots, m-1$ in one of k rounds $0, \ldots, k-1$. If exactly one robot and its corresponding drone arrive at a particular pad during a round, the package is transferred successfully. If a robot meets a drone expecting a different package, or if two robots or two drones attempt to use the same pad during the same round, the transfer fails and any workers choosing that pad in that round must try again at a later round. Each worker has no ability to communicate with other workers beyond being able to detect if a transfer it attempted failed or not.

It is clear that k must be at least $\lceil n/m \rceil$ for it to be possible to deliver all the packages even with perfect coordination. Show that this is almost sufficient, by giving an algorithm that, for any fixed $c > 0$, delivers all packages in $k = O((n/m) \log n)$ rounds with probability at least $1 - O(n^{-c})$.

[1] There is a very technical issue here involving how the bound on the number of bits interpreted. One extreme is that we have b bits of randomness and all 2^b possible bit strings are equally likely. The other extreme is that some algorithm generates a random variable or collection of random variables that collectively take on at most 2^b distinct possible values (and thus can be encoded in b bits), but there is no requirement that each possible value is equally likely. While the problem is solvable in either model, there are fewer details to worry about in the second model, so you should feel free to assume, for example, that you can encode a six-sided die in 3 bits by assigning 0 probability to 000 and 111 and 1/6 probability to each of the remaining six bit vectors.

Solution

We'll have a sequence of $p = O(\log n)$ phases of $\lceil 2n/m \rceil$ rounds each, and in phase use a hash function to assign each package to one of the $\ell = m\lceil 2n/m \rceil \geq 2n$ slots corresponding to a particular combination of pad and round within the phase.

A complication is that we have limited randomness, which is going to constrain what hash functions we can use. We'll pick an independent linear congruential hash function h_j for each phase j, which requires $O(\log t)$ bits of randomness per phase or $O(\log t \log n) = O(\log^2 t)$ bits of randomness across all phases. Tabulation hashing also works (we want the version that adds table elements mod ℓ rather than using XOR on bit vectors), but the bits per phase goes up to $O(\log^2 t)$ giving $O(\log^3 t)$ overall.

In each phase, a remaining robot or drone assigned tracking number t_i computes $h_j(t_i) \in [\ell]$ and goes to pad $h_j(t_i) \bmod modm$ during round $\lfloor h_j(t_i)/m \rfloor$ (numbering from 0 within the phase). Each pair of tracking numbers $t_i, t_{i'}$ produces a collision with probability $\Pr[h_j(t_i) = h_j(t_{i'})] \leq \frac{1}{\ell}$. Let X_j be the number of robots left after j phases; then $X_0 = n$ and

$$\mathrm{E}[X_{j+1} \mid X_j] \leq 2\binom{X_j}{2}\frac{1}{\ell} \leq \frac{X_j^2}{2n} \leq \frac{X_j}{2}.$$

Here the 2 at the start accounts for the fact that each collision may send up to two robots to the next phase, the $\binom{n}{2}$ counts all the pairs of robots, and the last step uses the bound $X_j \leq n$.

Iterating this inequality gives $\mathrm{E}[X_p] \leq n \cdot 2^{-p}$. Set $p = \lg n$ to get $\Pr[X_p > 0] \leq \mathrm{E}[X_p] \leq 2^{-c \lg n} = n^{-c}$. This gives the desired error probability in $c \lg n \lceil 2n/m \rceil = O(n \log n/m)$ rounds.

A.3.2 A linked list

It is well-known that linked lists are terrible data structures, with only $O(n)$ guarantees on search time in the worst case. However, for some distributions on elements, the average case might be better.

Suppose we are given a set of n elements $1, \ldots, n$, where at each step element i is supplied with probability p_i. Suppose that we insert each new element we see at the end of the list, until eventually all n elements appear in the list. We then sample a new element according to the given probability distribution and ask what its expected position in the list is.

For a uniform distribution, this won't help much: all $n!$ orderings will be equally likely, and the expected position of a random element is just

$\frac{n+1}{2} = \Theta(n)$, which is (up to constants) no better than the deterministic worst case. But for more skewed distributions we may hope that more probable elements get inserted early, meaning that the cost of searching for them will be less than for more improbable elements.

For each of the following distributions, compute a tight (big-Θ) asymptotic bound on the expected cost to search for a random element given by the distribution, assuming that we constructed the list as described above by repeatedly sampling from the same distribution.

1. A geometric distribution with $p_i \propto 2^{-i}$.
2. A **Zipf's law** distribution with $p_i \propto \frac{1}{i}$.

Solution

Let's do what we can for a generic distribution before getting into the details of each distribution individually.

Let A_{ij} be the indicator variable for the event that i appears in the list at or before the same position as j (the "at" part covers the case A_{ii}, which we take to be 1). Let D_i be the position of i in the list. Then $D_i = \sum_j A_{ji}$.

For $i \neq j$, we can compute $\mathrm{E}[A_{ij}]$ by looking at probability of seeing i conditioned on seeing i or j at a particular step. This gives

$$\begin{aligned}\mathrm{E}[A_{ij}] &= \frac{p_i}{p_i + p_j}, \\ \mathrm{E}[D_i] &= 1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j}.\end{aligned}$$

and thus

$$\mathrm{E}[\text{search cost}] = \sum_i p_i \mathrm{E}[D_i] = 1 + \sum_i p_i \sum_{j \neq i} \frac{p_j}{p_i + p_j}.$$

Let's see what happens to this for each of our given distributions.

1. When $p_i \propto 2^{-i}$, we have

$$p_i = \frac{2^{-i}}{1 - 2^{-n-1}} = \Theta\left(2^{-i}\right).$$

Even better, the denominators cancel while computing

$$\begin{aligned}\sum_{j\neq i}\frac{p_j}{p_i+p_j} &= \sum_{j\neq i}\frac{2^{-j}}{2^{-i}+2^{-j}}\\ &= \sum_{j<i}\frac{2^{-j}}{2^{-i}+2^{-j}} + \sum_{j>i}\frac{2^{-j}}{2^{-i}+2^{-j}}\\ &= \sum_{j<i}\frac{2^{-j}}{\Theta(2^{-j})} + \sum_{j>i}\frac{2^{-j}}{\Theta(2^{-i})}\\ &= \sum_{j<i}\Theta(1) + \sum_{j>i}\Theta(2^{i-j})\\ &= \Theta(i)+\Theta(1)\\ &= \Theta(i).\end{aligned}$$

We can now compute the expected search cost as

$$\sum_i p_i \operatorname{E}[D_i] = 1 + \sum_i \Theta(2^{-i}i) = \Theta(1).$$

2. In this case, we have

$$p_i = \frac{1/i}{H_n} = \frac{1}{iH_n}.$$

But the H_n's cancel out when we compute

$$\begin{aligned}\sum_{j\neq i}\frac{p_j}{p_i+p_j} &= \sum_{j\neq i}\frac{1/j}{1/i+1/j}\\ &= \sum_{j\neq i}\frac{i}{i+j}.\end{aligned}$$

We can use this to get an upper bound on the expected search cost

$$\begin{aligned}\sum_{i=1}^{n} p_i \operatorname{E}[D_i] &= \sum_{i=1}^{n} \frac{1}{iH_n}\left(1 + \sum_{j\neq i} \frac{i}{i+j}\right) \\ &< 1 + \frac{1}{H_n}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{1}{i+j} \\ &< 1 + \frac{1}{H_n}\sum_{k=1}^{2n} \frac{k-1}{k} \\ &< 1 + \frac{1}{H_n}\cdot 2n \\ &= O(n/\log n).\end{aligned}$$

To get a matching lower bound, notice that the best possible ordering places each i at position i, giving an expected search cost of exactly $\sum_i \frac{1}{iH_n}\cdot i = n/H_n = \Omega(n/\log n)$. Since whatever random ordering we end up with is at least this bad, this gives us the $\Theta(n/\log n)$ bound we are looking for.

A.4 Assignment 4, due Thursday 2024-03-28 at 23:59

A.4.1 Nearly orthogonal vectors

Given two vectors x and y, the angle θ_{xy} between them can be computed by the formula $\theta_{xy} = \cos^{-1}\frac{x\cdot y}{\|x\|\|y\|}$. If x and y are orthogonal, then $x\cdot y = 0$, and $\theta_{xy} = \pi/2$. Let's call two vectors x and y **nearly orthogonal** to within ϵ if $\pi/2 - \epsilon \le \theta_{xy} \le \pi/2 + \epsilon$.

Given an $n\times n$ matrix A, we can find all the pairs of rows A_i, A_j that are nearly orthogonal by computing θ_{A_i,A_j} for each pair and comparing it to $\pi/2 \pm \epsilon$. Unfortunately, this takes $\Theta(n^3)$ time if done in the obvious way. So let's allow a randomized approximation algorithm.

Claim: For any fixed ϵ with $0 < \epsilon \le \pi/4$, there is a randomized algorithm that takes as input an $n\times n$ matrix A and an error parameter $\delta > 0$, and returns in time $O(n^2 \log^c n \log^c(1/\delta))$, for some constant c, a list of nearly-orthogonal pairs of rows of A, such that with probability at least $1-\delta$, this list includes every pair of rows that are nearly orthogonal to within ϵ, and only pairs of rows that are nearly orthogonal to within 2ϵ.

Prove this claim.

Solution

This is a job for the Johnson-Lindenstrauss lemma (distributional version).

First normalize the rows of A so that each $\|A_i\| = 1$. This takes $O(n^2)$ time and doesn't affect the angles between rows.

Next, observe for any unit vectors x and y, $\|x - y\|_2$ is an increasing function of $\theta = \theta_{xy}$. We don't actually need anything more than this, but if we want to we can compute the distance exactly by constructing an isosceles triangle with x and y as the unit edges and chopping it in half: this gives $\|x - y\|_2 = 2 \sin \frac{\theta}{2}$. So if $\theta \leq \epsilon$, $\|x - y\|_2 \leq 2 \sin \frac{\epsilon}{2}$, and if $\theta \geq 2\epsilon$, $\|x - y\|_2 > 2 \sin \epsilon$.

Pick a threshold halfway between these two bounds. By choosing a small enough error term in Lemma 8.1.2, we can guarantee that any pair x, y that is nearly orthogonal to within ϵ has $\|f(x - y)\|_2$ close enough to $\|x - y\|^2$ to be within this threshold, and any pair x, y that is not nearly orthogonal to within 2ϵ doesn't. (We could calculate what the exact error bound we need for this, but since it's a constant, we don't care.) If we set the probability of exceeding the error bound for a single vector to $\delta / \binom{n}{2}$, this gives a probability that $\|A_i - A - j\|^2$ exceeds the relative error bound for any i, j of at most δ. To obtain $\delta / \binom{n}{2}$ probability of error, we need $k = O(\log(\delta / n^2)) = O(\log n + \log \delta)$.

So now we need to feed all of our rows to the Johnson-Lindenstrauss transform whose existence is given in the lemma and then check the threshold for each pair of rows in the output. Expanding out the steps, we must:

1. Compute an $n \times k$ transform matrix B implementing f. This takes $O(nk)$ time.

2. Compute AB. This takes $O(n^2 k)$ time using ordinary matrix multiplication.

3. For each pair of rows i and j, compute $\|(AB)_i - (AB)_j\|^2$ and check it against the thresholds. This takes $O(k)$ time per pair of rows for $O(n^2 k)$ time total.

Adding up all of these costs gives $O(n^2 k) = O(n(\log n + \log(1/\delta)))$ time, which is within the claimed bounds.

A.4.2 Boosting a random walk

Suppose you are made to participate in a fair ± 1 random walk, starting with $X_0 = k \leq n/2$, that ends if $X_t \geq n$ or $X_t \leq 0$. The twist is that once and

only once during the walk only you can push a "boost" button that replaces the normal transition rule $X_{t+1} = X_t \pm 1$ with a rule that doubles the current value: $X_{t+1} = 2X_t$. Naturally, your choice to use your boost can't depend on knowledge of the future: when choosing to set $X_{t+1} = 2X_t$, you only know the outcomes $X_1, \ldots, X_t$.

1. Suppose that your goal is to maximize the probability that you reach a state $X_t \geq n$ before you reach $X_t = 0$. What strategy should you use and what probability of reaching $X_t \geq n$ (as a function of k and n) does it give you?

2. Suppose instead that you want to *minimize* the probability that you reach a state $X_t \geq n$ before you reach reach 0, but you want to do this in a way that always uses the boost before you reach $X_t \geq n$ or $X_t = 0$, so that suspicious onlookers won't think you aren't trying. Now what strategy should you use, and what probability of reaching $X_t \geq n$ does it give you?

Solution

Let's formalize things a bit. Let $\{\Delta_t\}$ be the set of independent fair ± 1 increments, so that $X_{t+1} = X_t + \Delta_{t+1}$ if we don't use the boost. Then we can let $\mathcal{F}_t = \langle \Delta_1, \ldots, \Delta_t \rangle$ and make the time σ at which we use the boost a stopping time with respect to $\{\mathcal{F}_t\}$. This makes $[\sigma \leq t]$ measurable $\mathcal{F}_t$ for each t and gives $\mathrm{E}\left[\Delta_{t+1} \mid \mathcal{F}_t\right] = 0$.

We can now write the transition rule compactly as

$$X_{t+1} = 2^{[\sigma=t]} X_t + [\sigma \neq t]\Delta_{t+1},$$

where $[A]$ denotes the indicator variable for the event A.

Because of the boost, $\{X_t\}$ is not a martingale with respect to $\{\mathcal{F}_t\}$, but we can define a process that is.

Let $Y_t = 2^{[\sigma \geq t]} X_t$. The idea is that the first factor represents whether we still have a boost available to use or not.

Note that $[\sigma \geq t]$ starts at 1 when $t = 0$ and drops to 0 when we use the boost. We can write this as $[\sigma \geq t + 1] = [\sigma \geq t] - [\sigma = t]$. This lets us

compute

$$\begin{aligned}
\mathrm{E}\left[Y_{t+1} \mid \mathcal{F}_t\right] &= \mathrm{E}\left[2^{[\sigma \geq t+1]} X_{t+1} \;\middle|\; \mathcal{F}_t\right] \\
&= \mathrm{E}\left[2^{[\sigma \geq t+1]} \left(2^{[\sigma=t]} X_t + [\sigma \neq t] \Delta_{t+1}\right) \;\middle|\; \mathcal{F}_t\right] \\
&= 2^{[\sigma \geq t]-[\sigma=t]} \cdot 2^{[\sigma=t]} X_t + 2^{[\sigma \geq t+1]} [\sigma \neq t] \,\mathrm{E}\left[\Delta_{t+1} \mid \mathcal{F}_t\right] \\
&= 2^{[\sigma \geq t]} X_t \\
&= Y_t.
\end{aligned}$$

So $\{Y_t\}$ is a martingale with respect to $\{\mathcal{F}_t\}$.

(Here we could pull X_t, $2^{\sigma \geq t}$, $2^{\sigma \geq t+1} = 2^{[\sigma > t]}$, and $[\sigma \neq t]$ out of the conditional expectation because all these variables are measurable $\mathcal{F}_t$.)

Let $\tau = \min\{t \mid X_t \geq n \vee X_t = 0\}$. Then τ is finite with probability 1 (if nothing else, we eventually take n positive steps in a row), and $|Y_i| \leq 2n$, so we can apply the finite probability / bounded range version of the Optional Stopping Theorem to get

$$\begin{aligned}
\mathrm{E}\left[Y_0\right] &= \mathrm{E}\left[Y_\tau\right] \\
&= \mathrm{E}\left[Y_\tau \mid X_\tau \geq n\right] \Pr\left[X_\tau \geq n\right] + \mathrm{E}\left[Y_\tau \mid X_\tau = 0\right] \Pr\left[X_\tau = 0\right] \\
&= \mathrm{E}\left[Y_\tau \mid X_\tau \geq n\right] \Pr\left[X_\tau \geq n\right].
\end{aligned}$$

Since we know $\mathrm{E}\left[Y_0\right] = 2k$, this gives

$$\Pr\left[X_\tau \geq n\right] = \frac{2k}{\mathrm{E}\left[Y_\tau \mid X_\tau \geq n\right]}. \tag{A.4.1}$$

We should thus try to minimize or maximize $\mathrm{E}\left[Y_\tau \mid X_\tau \geq n\right]$ depending on whether we want to maximize or minimize $\Pr\left[X_\tau \geq n\right]$.

1. To minimize $\mathrm{E}\left[Y_t \mid X_\tau \geq n\right]$, observe that the smallest possible value for Y_τ given $X_\tau \geq n$ is $Y_\tau = X_\tau = n$. We can get this by using the boost immediately ($\sigma = 0$). Then we have a standard fair random walk starting from $X_1 = 2k$, which hits n before 0 with probability exactly $\frac{2k}{n}$.

   This is not necessarily the only good strategy, since any strategy that guarantees $\sigma < \tau$ and $X_\sigma \leq n/2$ will also get $\frac{2k}{n}$, but (A.4.1) tells us that no strategy can do better.

2. To maximize $\mathrm{E}\left[Y_\tau \mid X_\tau \geq n\right]$ subject to the requirement that we must use the boost, we want $\mathrm{E}\left[X_\tau \mid X_\tau \geq n\right]$ to be as large as possible. The

best we can hope to get is $X_\tau = 2(n-1)$, by using the boost on when $X_t = n-1$. But what if we don't reach $n-1$ and are forced to use the boost anyway?

Analyzing this using the Y_t martingale turns out to be tricky, and an easier approach is just to guess the worst strategy and show that it is in fact the worst. Since we must use the boost before τ, we are forced to use the boost if we reach 1 or $n-1$, since otherwise there is a possibility the process finishes before we use it. Let's suppose we adopt a maximum-delay strategy and set $\sigma = \min\{t \mid t \in \{1, n-1\}\}$.

Starting from any position x, this σ gives $\Pr[X_\tau = n]$ of $2/n$ if we reach 1 first and 1 if we reach $n-1$ first. Which we reach first is just the probability of hitting on or the other absorbing barrier in a simple random walk in $[1, n-1]$. So we can compute

$$\begin{aligned}
\Pr[X_\tau = n \mid X_t = x, t < \tau] &= \frac{x-1}{n-2} \cdot 1 + \left(1 - \frac{x-1}{n-2}\right) \cdot \frac{2}{n} \\
&= \frac{x-1}{n-2}\left(1 - \frac{2}{n}\right) + \frac{2}{n} \\
&= \frac{x-1}{n-2}\left(\frac{n-2}{n}\right) + \frac{2}{n} \\
&= \frac{x-1}{n} + \frac{2}{n} \\
&= \frac{x+1}{n}.
\end{aligned}$$

In particular, starting at $X_0 = k$ gives a probability of hitting the upper barrier of $\frac{k+1}{n}$.

Now suppose we consider some other strategy σ'. Like σ, σ' must use the boost at $X_t = 1$ or $X_t = n-1$. If $X_{\sigma'} = x$ for some x with $1 < x < n-1$, then $\Pr[X_\tau \geq n \mid X_\sigma = x] = \frac{2x}{n} > \frac{x+1}{n}$ since $x > 1$. So any other σ' gives a higher probability of $X_\tau \geq n$.

A.5 Assignment 5, due Thursday 2024-04-11 at 23:59

A.5.1 Relaxation time for Metropolis-Hastings

Suppose we have a lazy irreducible reversible Markov chain on n states with transition probabilities p_{ij}, uniform stationary distribution $\pi_i = 1/n$, and

relaxation time τ_2. Fix some function f with $1 \leq f(x) \leq r$ for all states x and apply Metropolis-Hastings to get a new Markov chain with transition probabilities $p'_{ij} = p_{ij} \min(1, f(j)/f(i))$ and stationary distribution $\pi'_i \propto f(i)$. Let τ'_2 be the relaxation time of this new chain.

Prove or disprove: Under these assumptions, there is an upper bound on τ'_2 that is polynomial in τ_2 and r.

Solution

We will prove this by showing that π'_u and p'_{uv} are both bounded relative to their unmodified versions by polynomial functions of r, then show this leads to at most polynomial blowup going from τ_2 to the conductance Φ of the original chain to the conductance Φ' of the modified chain and finally to τ'_2.

Let n be the number of states. For π'_u, we have

$$\pi'_u = \frac{f(u)}{\sum_v f(v)} \geq \frac{1}{(n-1)r+1} > \frac{1}{nr} = \frac{1}{r}\pi_u$$

and in the other direction

$$\pi'_u = \frac{f(u)}{\sum_v f(v)} \leq \frac{r}{(n-1)+r} < \frac{r}{n} = r\pi_u.$$

For $p'_{uv} = p_{uv} \min(1, f(v)/f(u))$, we get $\frac{1}{r}p_{uv} \leq p'_{uv} \leq p_{uv}$.

So now let's look at some set of states S and try to bound $\Phi'(S)$.

$$\begin{aligned}
\Phi'(S) &= \frac{\sum_{u\in S, u\notin S} \pi'_u p'_{uv}}{\pi'(S)} \\
&> \frac{\sum_{u\in S, u\notin S} \frac{1}{r}\pi_u \cdot \frac{1}{r}p_{uv}}{r\pi(S)} \\
&= \frac{1}{r^3}\frac{\sum_{u\in S, u\notin S} \pi_u p_{uv}}{\pi(S)} \\
&= \frac{1}{r^3}\Phi(S). \qquad \text{(A.5.1)}
\end{aligned}$$

We'd like to use this to bound $\Phi' = \min_{0<\pi'(S)\leq 1/2} \Phi'(S)$ in terms of r and $\Phi = \min_{0<\pi(S)\leq 1/2} \Phi(S)$, but there is a complication: there may be sets S with $\pi'(S) \leq 1/2$ but $\pi(S) > 1/2$ that are included in the computation of Φ' but not Φ. For these sets we will need to take advantage of reversibility.

Suppose $\pi'(S) \leq 1/2$, $\pi(S) > 1/2$. Let $T = \overline{S}$. Then $\pi(T) = 1 - \pi(S) < 1/2$, so $\Phi(T) \leq \Phi$. Now we can argue

$$\begin{aligned}
\Phi'(S) &= \frac{\sum_{u \in S, v \in T} \pi'_u p'_{uv}}{\pi'(S)} \\
&= \frac{\sum_{v \in T, u \in S} \pi'_v p'_{vu}}{\pi'(S)} \\
&= \Phi'(T) \frac{\pi'(T)}{\pi'(S)}. \\
&> \frac{1}{r^3} \Phi(T) \frac{\pi'(T)}{\pi'(S)}. \\
&> \frac{1}{r^3} \Phi \cdot \frac{1/2}{1/2} \\
&= \frac{1}{r^3} \Phi. \qquad \text{(A.5.2)}
\end{aligned}$$

So given S with $0 < \pi'(S) \leq 1/2$, (A.5.1) shows $\Phi'(S) \geq r^{-3}\Phi$ when $\pi(S) \leq 1/2$ and (A.5.2) shows $\Phi'(S) \geq r^{-3}\Phi$ when $\pi(S) > 1/2$. In either case we get $\Phi' = \min_{0 < \pi'(S) \leq 1/2} \Phi'(S) \geq r^{-3}\Phi$.

From (10.6.4), we have $\frac{1}{2\Phi} \leq \tau_2$, which gives $\Phi \geq \frac{1}{2\tau_2}$. But then we can apply the other direction of (10.6.4) to get

$$\tau'_2 \leq \frac{2}{(\Phi')^2} < \frac{2}{(r^{-3}\Phi)^2} \leq \frac{2r^6}{(1/(2\tau_2))^2} = 8r^6\tau_2^2,$$

which is polynomial in τ_2 and r.

Though it doesn't affect the solution to the problem, it is probably worth noting here that there may be a much more efficient way to sample proportional to f, which is to sample a point uniformly (in time $O(\tau_2 \log n/\epsilon)$) and then apply rejection sampling to adjust the probabilities. This yields a sample in $O(r\tau_2 \log n/\epsilon)$ in the worst case where f is 1 for most points and so we only return a sample with probability $1/r$. So we may wish to reserve Metropolis-Hastings for situations where f has a much larger range (making rejection sampling too costly) and we can show a tighter bound on convergence time in some other way.

A.5.2 A constrained random walk

Let's imagine we want to sample n-bit strings with no consecutive pairs of ones. So 00101 is acceptable but 01101 is not.

It's possible to do this efficiently and exactly by taking advantage of the connection of these strings to Fibonacci numbers, but for some unexplained reason we decide to do this using Markov Chain Monte Carlo instead. Given an initial string x, pick an index i and bit b uniformly at random, and set x_i to b unless this would produce two consecutive ones. It is not hard to show that this gives an irreducible aperiodic Markov chain that converges to a uniform stationary distribution.

Show that this chain has the rapid mixing property, which means that $t_{\text{mix}}(\epsilon)$ is polylogarithmic in the number of states N and the total variation distance bound ϵ.

Solution

First observe that the number of possible states N is at least $2^{\lceil n/2 \rceil}$, the number of strings with zeros in all odd-numbered positions.[2] So if we can get a mixing time that is polynomial in n, it will be polylogarithmic in n.

Coupling and canonical paths are both options for showing such a bound.

One possible coupling is to apply the same update rule to both the X and Y processes: Pick an index i and a bit i and update each of X_i and Y_i to equal b if permitted by the no consecutive ones rule. We can then track how the Hamming distance between X and Y evolves over time.

Let $S^t = \{i \mid X_i^t \neq Y_i^t\}$ and let $H^t = |S^t|$. Then H^t can change in two ways:

- If we pick some $i \in S$, then H always drops by 1 if $b = 0$, and drops by 1 if $b = 1$ and neither $i - 1$ nor $i + 1$ is in S.
- If we pick some $i \notin S$, then H stays the same if $i - 1$ and $i + 1$ are both not in S and goes up by 1 if either is in S.

Partition S into segments $S_1, \ldots, S_k$ where each S_k is a maximal sequence of consecutive indices.

[2] This is a rather crude estimate that is convenient mostly because itit is instantly recognizable as exponential in n. As hinted at in the problem statement, we can compute N exactly using Fibonacci numbers. Each acceptable string of length 2 or more either starts with 0 followed by an acceptable string of length $n - 1$, or starts with 10 followed by an acceptable string of length $n - 2$. This gives a recurrence $N(n) = N(n - 1) + N(n - 2$ with $N(0) = 1$ and $N(1) = 2$. This is exactly the recurrence for Fibonacci numbers shifted so that $N(n) = F_{n+2}$, which is not surprising when we recall that Pingala invented Fibonacci numbers precisely to count sequences of syllables of length 1 or 2 adding up to a given total.

If $|S_i| = 1$, then the index in S_i contributes a $\frac{1}{n}$ chance of lowering H, and the adjacent indices together contribute at most a $\frac{1}{n}$ chance of increasing H.

If $|S_i| = k > 1$, then each index in S_i contributes a $\frac{1}{2n}$ chance of lowering H, for a total of $\frac{k}{2n} \geq \frac{1}{n}$, and the adjacent indices together contribute at most a $\frac{1}{n}$ chance of increasing H.

In either case we have $\mathrm{E}\left[H^{t+1} \mid X^t, Y^t\right] \leq H^t$, with at least a $\frac{2}{n}$ chance that H changes if $H > 0$. We also have that H never exceeds n and never changes once it reaches 0. So we can bound H by a fair random walk H' with a reflecting barrier at n and absorbing barrier at 0, that only takes a step when H does. This shows that H reaches 0 in $O(n^3)$ steps on average.

For canonical paths, given two states x and y that differ in k places, we'll define a path γ_{xy} of length k that fixes these bits one at a time. We can't do strict left-to-right bit fixing, because this may create an intermediate state that has two consecutive ones. But we can adjust the order a little to get around this: at each step, change the leftmost unfixed bit that can be changed without violating the no-consecutive ones rule.

A step in such a path changes some string of the form $y_1 \dots y_\ell r s x_{\ell+3} \dots x_n$ to $y_1 \dots y_\ell r' s' x_{\ell+3} \dots x_n$, where the $rs \to r's'$ change is one of $00 \to 10$, $01 \to 00$, or $10 \to 00$. To reconstruct x and y from this information, it is enough to provide ℓ ($n-2$ choices), a string $x_1 \dots x_{\ell+1} 0 y_{\ell+3} \dots y_n$ with no consecutive ones ($< N$ choices), and the values of $x_{\ell+2}$ and $y_{\ell+2}$ (4 choices). This gives $< 4nN$ paths crossing each edge, so the congestion ρ is

$$\max_{uv \in E} \frac{1}{\pi_u p_{uv}} \sum_{\gamma_{xy} \ni uv} \pi_x \pi_y < \frac{1}{N^{-1}(1/(2n))} \cdot 4nN \cdot N^{-2} = 8n^2.$$

We thus get $\tau_2 \leq 8\rho^2 = O(n^4)$, with a hideous constant swept behind the big O for decency's sake. This again is polylogarithmic in N, so we're done.

A.6 Assignment 6, due Thursday 2024-04-25 at 23:59

A.6.1 The power of intransigence

Let G be a connected graph on n vertices labeled $1, \dots, n$, and imagine each vertex v in G represents a person who has an opinion $X_v^t \in \{0, 1\}$ at each time t. This opinion is initially arbitrary, but may evolve over time to match the opinions of v's neighbors.

Specifically, at each step, we pick one of the m edges in the graph uniformly at random, and then choose a random orientation of that edge to get an ordered pair of adjacent vertices ij, with each possible pair chosen with probability $\frac{1}{2m}$. Most of the time, this results in j adopting i's opinion, so that $X_j^{t+1} = X_i^t$, while $X_k^{t+1} = X_k^t$ for all $k \neq j$.

But not if $j = 1$. Vertex 1 is stubborn: headstrong, willful, convinced in its bones that it is right, holding a graduate degree in obstinacy in one hand and a thesaurus opened to "stubborn" in the other to back up its unyielding pig-headed beliefs. If we are unlucky enough to pick $j = 1$, then nothing changes and $X^{t+1} = X^t$.

Running this process long enough will eventually converge to a state where every other vertex gives up and agrees with vertex 1. There is even a constant c such that it reaches this state in $\Theta(n^c)$ steps on average with the worst possible G and the worst possible X^0.

Find c, and prove that this bound holds.

Solution

We'll show $c = 4$.

For the lower bound, we just need one bad family of graphs and starting configurations that gives $\Omega(n^4)$ expected convergence time. One choice is a lollipop graph, consisting of a handle that is a path of $n/2$ vertices and a sucker that is $K_{n/2}$, plus an edge linking one end of the path to one vertex in the clique.

Let's make the stubborn node be a member of the clique with initial value 0, and start in a configuration where the most distant $n/4$ nodes in the path have 1 and everybody else has 0. Let X_i^t be the value of node i at time t and let $Z_t = \sum_i X^t$. Then as long as all the clique nodes have $X_i^t = 0$ and some path node has $X_i^t = 1$, we have a configuration where the 1 nodes are exactly the Z_t most distant nodes on the path. So $Z^{t+1} = Z_t$ only if we pick the unique $0 - 1$ edge, and we have $Z^{t+1} = Z_t \pm 1$ with probability $\frac{1}{2m}$ each.

This give an unbiased random walk that takes steps with probability $\frac{1}{m}$. It takes $(n/4)^2$ steps for this to reach $Z_t = 0$ or $Z_t = n/2$, which is a lower bound on the time for the process as a whole to converge. The waiting time for each random walk step is m, so the total expected time to reach $Z_t = 0$ or $Z_t = n/2$ is $m(n/4)^2$ by Wald's Equation. The clique makes $m = \Omega(n^2)$, so the expected time to converge is $\Omega(n^4)$.

In the other direction, we will show that any graph reaches an all-equal state from any initial state in time $O(n^4)$. For this argument it's convenient to assume that the stubborn node starts with 1, so we finish when $Z_t = n$.

Fix some configuration X^t, and let ℓ be the number of $0-1$ edges in X^t. Let $\mathcal{F}_t = \langle X^0, X^1, \ldots, X^t \rangle$ be the σ-algebra generated by the trajectory of the system up until time t. Then $\Pr\left[Z^{t+1} - Z_t = 1\right] \mathcal{F}_t = \frac{\ell}{2m}$ and $\Pr\left[Z^{t+1} - Z_t = -1\right] \mathcal{F}_t \leq \frac{\ell}{2m}$ (because edges incident to the stubborn vertex don't contribute). The rest of the time $Z^{t+1} = Z_t$.

This gives a ± 1 random walk that may be biased toward $+1$ and that takes steps only some of the time. There are two mostly equivalent ways to bound the convergence time of this process: one uses an explicit potential function, and the other reduces to known results about random walks via a coupling argument. We'll describe both below, since either gives a reasonable solution to this part of the problem.

Potential function argument Consider the function Z_t^2. The expected change in this quantity is

$$\begin{aligned}
\mathrm{E}\left[Z_{t+1}^2 - Z_t^2 \;\middle|\; \mathcal{F}_t\right] &\geq (2Z_t + 1)\Pr\left[Z_{t+1} = Z_t + 1 \mid \mathcal{F} + t\right] \\
&\quad + (-2Z_t + 1)\Pr\left[Z_{t+1} = Z_t - 1 \mid \mathcal{F} + t\right] \\
&\geq (2Z_t + 1)\frac{\ell}{2m} + (-2Z_t + 1)\frac{\ell}{2m} \\
&= \frac{\ell}{m} \\
&\geq \frac{1}{m}.
\end{aligned}$$

(The second step uses $Z_t \geq 1$, as enforced by the stubborn vertex.)

If we define $Y_t = Z_t^2 - \frac{t}{m}$, then Y_t is a submartingale, since the expected increase in Z_t^2 always pays for the increase in $\frac{t}{m}$. So at any stopping time τ that satisfies the conditions of the Optional Stopping Theorem for $\{Y_t\}$, $\mathrm{E}\left[Y_\tau\right] \geq \mathrm{E}\left[Y_0\right]$.

Let τ be the first time at which $Z_\tau = n$. Then $Y_\tau = Z_\tau^2 - \frac{\tau}{m} = n^2 - \frac{\tau}{m}$. For $t \leq \tau$, Y_t and τ satisfy the bounded increments and finite expected time conditions, because Y_t can't change by more than $2n + \frac{1}{m}$ in this range and τ has finite expectation by the usual argument for Markov chain hitting times. So $\mathrm{E}\left[Y_\tau\right] = n^2 - \frac{1}{m}\mathrm{E}\left[\tau\right] \geq Y_0 = Z_0^2 \geq 0$. Solve to get $\mathrm{E}\left[\tau\right] \leq mn^2 = O(n^4)$.

Coupling argument An alternative argument couples Z^t with an unbiased random walk W_t with a reflecting barrier at 0, that takes steps at random times $T_1, T_2, \ldots$, where $\mathrm{E}\left[T_{i+1} - T_i\right] \leq m$. The idea is to move W_t only when we pick a $0-1$ edge n the X^t process, and argue that we can always keep $Z_t \geq W_t$ by incrementing W_t whenever we increment Z_t and decrementing W_t

even if Z_t doesn't go down because of stubbornness. So when W_t reaches n after at most n^2m expected steps (Wald's Equation again!), Z_t must already have reached n.

(This is pretty much the same argument as the potential function argument, except that we hide the potential function inside the known n^2 bound for the unbiased random walk. I think the coupling argument is more satisfying in some ways because it more intuitively describes what the Z_t process is doing. However, at least as written above, it skips over justifying some details that don't come up with the potential function, so there is also more room for error.)

A.6.2 Almost Markov

Let G be a d-regular graph on n vertices. Assign a label $\ell(v) \in \{0, 1\}$ to each vertex. Let $V_0V_1V_2\ldots$ be a random walk on G starting at some initial fixed vertex v_0, where each V_{t+1} is chosen uniformly among the d neighbors of V_t, and let $\ell(V_0), \ell(V_1), \ell(V_2), \ldots$ be the sequence of vertex labels observed during this random walk.

Call the sequence $\ell(V_0), \ell(V_1), \ell(V_2), \ldots$ **almost Markov** with error ϵ if, for all $t \geq 0$, $1/2 - \epsilon \leq \mathrm{E}\left[\ell(V_{t+1}) \mid \ell(V_0), \ldots, \ell(V_t)\right] \leq 1/2 + \epsilon$.

Show that, for any fixed $\epsilon > 0$, there is some minimum degree d_ϵ, such that for any d-regular graph G on n vertices, it is possible to compute a labeling on G in expected time polynomial in n and d, such that a random walk on G starting from any vertex produces a sequence of labels $\ell(V_0), \ell(V_1), \ell(V_2), \ldots$ that is almost Markov with error ϵ.

Solution

Suppose we label each vertex independently and uniformly at random. For each vertex v, let $N(v)$ be the set of all nodes adjacent to v, let $L_v = \frac{1}{d}\sum_{u \in N(v)} \ell(v)$ be the probability that taking a step starting at V lands on a node labeled with a 1, and let A_v be the event $|L_v - 1/2| > \epsilon$.

Observe that if none of the events A_v occurs, we get the almost-Markov property because the bound on $\mathrm{E}\left[\ell(V_{t+1}\right]$ will hold conditioned on any previous vertex V_t. We will use the Lovász Local Lemma to show that a labeling exists that makes none of these occur, and then use Moser-Tardos to find one.

With a bit of tinkering, we can write $L_v - 1/2$ as the sum of d independent

$\pm\frac{1}{2d}$ random variables, so Hoeffding's inequality says

$$p = \Pr[A_v] \leq \Pr[|L_v - 1/2| \geq \epsilon] \leq 2e^{-\epsilon^2/2d(1/d^2)} = 2e^{-d\epsilon^2}.$$

Let $\Gamma(A_v) = \{A_u \mid d(v,u) = 2\}$. Then A_v is independent of any event $A_u \notin \Gamma(A_v) \cup \{A_v\}$, because only events in this set depend on vertices that are adjacent to v, which are the only ones that affect A_v. Because the graph is d-regular, there are at most $d(d-1)$ vertices at distance 2 from v, and thus at most $d(d-1)$ events in $\Gamma(A_v)$.

The Lovász Local Lemma applies if $ep(d(d-1)+1) < 1$, which holds if $e \cdot 2e^{-d\epsilon^2} \cdot (d(d-1)+1) < 1$. For any fixed ϵ, the expression on the left goes to zero in the limit as d increases, so there exists some value d_ϵ such that the bound holds for all $d \geq d_\epsilon$. If the bound does hold, LLL says that there exists a labeling that makes none of the A_v happen.

Using the Moser-Tardos algorithm [MT10], we can find such a labeling in $\frac{n}{d(d-1)}$ resamplings on average. The efficient way to manage the resamplings is to compute all the L_v values initially in $O(nd)$ time, and then pay $O(d^2)$ time for each resampling to update the values and check any that have changed. If we do this, the initial setup cost dominates on average, giving an overall expected $O(nd)$ total time, which is about the best we could reasonably hope for given the size of the input, and which is polynomial in n and d as required.

Appendix B

Sample assignments from Spring 2023

B.1 Assignment 1, due Thursday 2023-02-16 at 23:59

B.1.1 Hashing without counting

Suppose we are building a hash table with separate chaining, which is implemented as an array of m positions $A[0], A[1], \ldots, A[m-1]$, each of which may hold zero or more elements in some secondary data structure like a variable-length array or linked list (see §7.1). We'll abstract away from the actual hash function and assume that when a new element is inserted, its position is chosen independently and uniformly at random.

Normally we would keep track of the number of elements n inserted into the table and grow the table by some constant factor when the load factor $\alpha = n/m$ gets too big. But this is work and we are lazy. So instead we adopt the following heuristic: grow the table as soon as some element is inserted into position 0. We'd like to make sure that this heuristic doesn't produce any bad outcomes, by looking at how crowded the positions in the table are at the moment after this element is placed in $A[0]$, just before expanding the table.

Specifically, for this configuration:

1. Compute the exact expected load factor $\alpha = n/m$ as a function of m.

2. Compute the best high-probability asymptotic bound you can for the maximum number of elements in any table position as a function of

m. This means that you should find a function $f(m)$ such that for all $c > 0$, the maximum number of elements is at most $O(f(m))$ with probability at least $1 - m^{-c}$, where the constant in the asymptotic bound may depend on c.

Solution

1. Let X be the number of elements inserted into the table. Then X counts the number of insertions up to and including the first insertion to position 0, making it a geometric random variable with parameter $p = 1/m$ (see §3.6.2). This gives $E[X] = m$.

   The expected load factor is $E[X/m] = 1$.

2. Pick some $i \neq 0$, and let X_i be the number of elements in position i.

   Let's calculate $\Pr[X_i \geq k]$. If $k = 0$, this is 1. Otherwise, when we insert the first element, there is a $1/n$ chance it lands in position i, and a $1/n$ chance it lands in position 0. The rest of the time it lands in a position $j \notin \{0, i\}$ and has no effect on X_i.

   So we have, for $k > 0$,

$$\begin{aligned}\Pr[X_i \geq k] = &\frac{1}{n}\Pr[X_i \geq k \mid \text{insert } 0] \\ &+ \frac{1}{n}\Pr[X_i \geq k \mid \text{insert } i] \\ &+ \frac{n-2}{n}\Pr[X_i \geq k \mid \text{insert elsewhere}]\end{aligned}$$

   which gives

$$\begin{aligned}\Pr[X_i \geq k] = &\frac{1}{2}\Pr[X_i \geq k \mid \text{insert } 0] \\ &+ \frac{1}{2}\Pr[X_i \geq k \mid \text{insert } i].\end{aligned}$$

   The first branch is 0 when $k > 0$. For the second branch, observe that $\Pr[X_i \geq k \mid \text{insert } i] = \Pr[X_i \geq k - 1]$, since we already have one element and now we are asking if we can get $k-1$ more using the same process that generates X_i. This gives a recurrence:

$$\Pr[X_i \geq k] = \begin{cases} 1 & k = 0 \\ \frac{1}{2}\Pr[X_i \geq k-1] & k > 0 \end{cases}$$

which has the solution $\Pr[X_i \geq k] = 2^{-k}$.

Now use the union bound to compute, for $k > 1$:

$$\begin{aligned}\Pr[\max X_i \geq k] &= \Pr[\exists i : X_i \geq k] \\ &\leq \sum_{i=1}^{n-1} \Pr[X_i \geq k] \\ &= (n-1)2^{-k}.\end{aligned}$$

Pick any $c > 0$. Let $k = (c+1)\lg n$. Then

$$\begin{aligned}\Pr[\max X_i \geq k] &\leq (n-1)2^{-(c+1)\lg n} \\ &= (n-1)n^{-c-1} \\ &< n^{-c},\end{aligned}$$

from which it follows that

$$\Pr[\max X_i < (c+1)\lg n)] \geq 1 - n^{-c}.$$

Since this makes $\max X_i = O(\log n)$ with probability $1 - n^{-c}$ for any fixed $c > 0$, we get a high-probability bound of $O(\log n)$.

Though this is not required by the problem, we can see that it is not possible to do better than $O(\log m)$ by looking at the distribution for a single bin. We have $\Pr[\max X_i \geq k] \geq \Pr[X_1 \geq k] = 2^{-k}$, so $\Pr[\max X_i \geq c \lg m] \geq m^{-c}$ for any fixed c.

B.1.2 Permutation routing on an incomplete network

Suppose we have a network of $n = 2d$ senders and $n = 2d$ receivers, where each sender has outgoing links to exactly d distinct receivers. Other than this information, the structure of the graph is unconstrained.

Let the senders be labeled $1 \ldots n$ and let the receivers be labeled $\pi_1, \ldots, \pi_n$ where π is a permutation of $1 \ldots n$ chosen uniformly at random. Call a sender i "successful" if there is a link from sender i to the receiver who is assigned label i. Let S be the number of successful senders. Then S is a random variable that depends on the choice of π. (Figure B.1 shows an example.)

1. What is the exact value of $\mathrm{E}[S]$ as a function of n?

2. What are the smallest and largest possible exact values of $\mathrm{Var}[S]$ as a function of n?

3. Show that for any choice of graph that satisfies the constraints above, at least $\Omega(n)$ senders are successful with probability at least $1 - o(1)$.

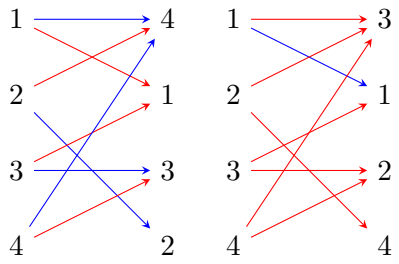


Figure B.1: An incomplete network with $n = 2d = 4$. In the left copy, the random permutation makes all four senders successful, giving $S = 4$. In the right copy, a different random permutation makes only sender 1 successful, giving $S = 1$.

Solution

1. Let X_i be the indicator variable for the event that sender i is successful, so that $S = \sum_{i=1}^{n} X_i$. Each sender i is successful if and only if i is the label of one of i's d neighbors. This occurs with probability $d/n = 1/2$, so $\mathrm{E}[X_i] = 1/2$ and $\mathrm{E}[S] = n/2$.

2. We have $\mathrm{Var}[S] = \sum_i \mathrm{Var}[X_i] + \sum_{i \neq j} \mathrm{Cov}[X_i, Y_i]$. Since the X_i are fair coin-flips we get $\mathrm{Var}[X_i] = 1/4$.

   To bound $\mathrm{Cov}[X_i, Y_i]$ let δ_i and δ_j be the neighborhoods of senders i and j, and let $c_{ij} = |\delta_i \cap \delta_j|$ be the number of neighbors these senders have in common. Then

$$\begin{aligned}
\mathrm{E}[X_i X_j] &= \Pr[i \in \delta_i \wedge j \in \delta_j] \\
&= \Pr[i \in \delta_i \cap \delta_j] \Pr[j \in \delta_j \mid i \in \delta_i \cap \delta_j] \\
&\quad + \Pr[i \in \delta_i \setminus \delta_j] \Pr[j \in \delta_j \mid i \setminus \delta_i \cap \delta_j] \\
&= \frac{c_{ij}}{n} \cdot \frac{d-1}{n-1} + \frac{d - c_{ij}}{n} \cdot \frac{d}{n-1} \\
&= \frac{c_{ij}(d-1) + (d - c_{ij})d}{n(n-1)} \\
&= \frac{c_{ij}d - c_{ij} + d^2 - c_{ij}d}{n(n-1)} \\
&= \frac{d^2 - c_{ij}}{n(n-1)}.
\end{aligned}$$

This gives

$$\begin{aligned}\operatorname{Cov}[X_i, X_j] &= \operatorname{E}[X_i X_j] - \operatorname{E}[X_i]\operatorname{E}[X_j] \\ &= \frac{d^2 - c_{ij}}{n(n-1)} - \frac{d^2}{n^2} \\ &= \frac{d^2 n - c_{ij} n - d^2(n-1)}{n^2(n-1)} \\ &= \frac{d^2 - c_{ij} n}{n^2(n-1)} \\ &= \frac{1/4 - c_{ij}/n}{n-1}.\end{aligned}$$

Summing over all $n(n-1)$ pairs $i \neq j$ gives

$$\begin{aligned}\operatorname{Var}[S] &= \sum_i \operatorname{Var}[X_i] + \sum_{i \neq j} \operatorname{Cov}[X_i, X_j] \\ &= \frac{n}{4} + \frac{n}{4} - \sum_{i \neq j} \frac{c_{ij}}{n(n-1)}. \\ &= \frac{n}{2} - \frac{1}{n(n-1)} \sum_{i \neq j} c_{ij}.\end{aligned}$$

To minimize the variance, we want to make $\sum_{i \neq j} c_{ij}$ as large as possible. We can do this by routing all n senders to the same $d = n/2$ receivers, giving $c_{ij} = n/2$ for all $i \neq j$ and thus $\operatorname{Var}[S] = 0$. This is not surprising because in this case, all and only those senders whose ids appear among the d favored receivers will be successful, making S a constant.

For the upper bound, we can get crude bound by observing that $c_{ij} \geq 0$ gives $\operatorname{Var}[S] \leq n/2$. But in general there is no graph that actually produces $c_{ij} = 0$ for all $i \neq j$. Instead, we need to look at minimizing $\sum c_{ij}$ subject to the constraint that there are nd edges.

Let e_{ijk} be 1 if senders i and j share an edge to receiver k and 0 otherwise. Then $c_{ij} = \sum_k e_{ijk}$. But if we write d_k for the degree of receiver k, we also have that $\sum_{i \neq j} e_{ijk} = d_k(d_k - 1)$, since each ordered pair of distinct neighbors of k contributes one nonzero e_{ijk}. It follows

that

$$\begin{aligned}\sum_{i\neq j} c_{ij} &= \sum_{i\neq j}\sum_{k} e_{ijk} \\ &= \sum_{k} d_k(d_k-1).\end{aligned}$$

We have the constraint $\sum_k d_k = dn$; subject to this constraint, $\sum_k d_k(d_k-1)$ is minimized when all $d_k = d$. This can be achieved by routing each sender i to receivers $i, i+1, \ldots i+(d-1) \pmod n$ or by routing all of the first d senders to all of the first d receivers and similarly for the second d senders and receivers.

This gives

$$\begin{aligned}\operatorname{Var}[S] &= \frac{n}{2} - \frac{1}{n(n-1)}\sum_{i\neq j} c_{ij} \\ &\leq \frac{n}{2} - \frac{1}{n(n-1)}\cdot nd(d-1) \\ &= \frac{n}{2} - \frac{d(d-1)}{n-1} \\ &= \frac{n}{2}\left(1 - \frac{n/2-1}{n-1}\right) \\ &= \frac{n}{2}\cdot\frac{2n-2-n+2}{2(n-1} \\ &= \frac{n^2}{4(n-1)}.\end{aligned}$$

As a check, we can observe that when $n = 2$, this gives $\operatorname{Var}[S] = 1 = n/2$, matching the crude bound because in this case we can in fact guarantee no overlap between senders' neighborhoods.

3. Since $\operatorname{Var}[S] \leq n/2$, we can use Chebyshev's inequality to compute

$$\begin{aligned}\Pr[S \leq n/4] &= \Pr[S - \operatorname{E}[S] \leq -n/4] \\ &\leq \Pr[|S - \operatorname{E}[S]| \geq n/4] \\ &\leq \frac{n/2}{(n/4)^2} \\ &= 1/n.\end{aligned}$$

So our probability of getting at least $n/4 = \Omega(n)$ successful senders is at least $1 - 1/n = 1 - o(1)$.

A tighter bound can be obtained using Azuma's inequality with an appropriate exposure martingale. Let $\mathcal{F}_t$ be the σ-algebra generated by knowing which receiver is assigned label j for all $j \leq t$. Let $Y_t = \mathrm{E}[S \mid \mathcal{F}_t]$.

To apply Azuma's inequality, we need a bound on $|Y_{t+1} - Y_T|$. Let's look at $\mathrm{E}[X_i \mid \mathcal{F}_t]$ and how it changes with t.

(a) If $i \leq t$, then the target for sender i is already known, and X_i is either 0 or 1. The change $\mathrm{E}[X_i \mid \mathcal{F}_{t+1}] - \mathrm{E}[X_I \mid \mathcal{F}_{t+1}]$ is exactly 0 in this case.

(b) If $i = t + 1$, then $\mathcal{F}_{t+1}$ determines x_i. Whatever the previous conditional expectation, the new conditional expectation is either 0 or 1. This gives a change of at most ± 1.

(c) If $i > t + 1$, then there are $n - t$ possible places to put label i, of which d_i are connected to sender i for some $0 \leq d_i \leq d$. The value of d_i depends on how many receivers connected to sender i have known labels in $\mathcal{F}_t$.

In this case we have

$$\begin{aligned}
\mathrm{E}[X_i \mid \mathcal{F}_{t+1}] - \mathrm{E}[X_i \mid \mathcal{F}_t] &\leq \frac{d_i}{n-t-1} - \frac{d_i}{n-t} \\
&= \frac{d_i}{(n-t)(n-t-1)} \\
&\leq \frac{n-t}{(n-t)(n-t-1)}. \\
&\leq \frac{1}{n-t-1}.
\end{aligned}$$

In the other direction,

$$\begin{aligned}
\mathrm{E}[X_i \mid \mathcal{F}_{t+1}] - \mathrm{E}[X_i \mid \mathcal{F}_t] &\geq \frac{d_i - 1}{n-t-1} - \frac{d_i}{n-t} \\
&= -\frac{n-t-d_i}{(n-t)(n-t-1)} \\
&\geq -\frac{n-t}{(n-t)(n-t-1)}, \\
&\geq -\frac{1}{n-t-1}.
\end{aligned}$$

In either case the absolute value is bounded by $\frac{1}{n-t-1}$. There are $n-t-1$ values of i in this range, so the sum for all of them is at most ± 1.

Combining the last two cases, we have $|\mathrm{E}[S \mid \mathcal{F}_{t+1}] - \mathrm{E}[S \mid \mathcal{F}_t]| \leq 2$. So Azuma's inequality gives $\Pr[S - n/2 \geq t] \leq e^{-t^2/8n}$ and similarly $\Pr[S - n/2 \leq t] \leq e^{-t^2/8n}$. The constant in the denominator could possibly be improved a bit with a better analysis, but even as is we get that $S = n/2 \pm \Theta(\sqrt{n \log n})$ with high probability and $S \geq n/4 = \Omega(n)$ with all but exponentially small probability.

B.2 Assignment 2, due Thursday 2023-03-30 at 23:59

B.2.1 Some streaming data structures

Suppose you are given a stream $x_1, x_2, \ldots, x_{2n}$ of $2n$ integer values in the range $0 \ldots n-1$. You are presented with these values one at a time. Upon receiving each value, you may update a data structure of your choosing that fits in $O(\log n)$ bits of space. You have no other storage.

1. Give an algorithm for sampling a single value with probability proportional to the number of times it occurs in the stream, returning this value once all elements of the stream have been processed.

   For example, given a stream $1, 0, 1, 2, 1, 1$, your algorithm should return 0 or 2 with probability 1/6 each, and return 1 with probability 2/3.

2. Give a Las Vegas algorithm that returns a value that occurs at least twice in the stream with probability $\Omega(1)$. If it doesn't succeed, it should return $\perp$ to indicate failure. (This algorithm should never return a value that occurs less than twice.)

In both cases, prove the correctness of your algorithm. You may assume that n is known to the algorithm.

Solution

1. There are two straightforward ways to do this using $O(\log n)$ bits:

   (a) The easiest is to sample the position of the desired element ahead of time, then count down until we hit it. This requires $2n$ possible

states for the countdown timer plus n possible states for the stored value, for a total of $\lceil \lg 3n \rceil = O(\log n)$ bits.

(b) Alternatively, we could store a value while counting how many values we've seen, and replace the stored value with the k-th value with probability $1/k$.[1] A straightforward induction argument shows that this makes the stored value uniform across all the values seen so far. We need $\lceil \lg 2n \rceil$ bits for the counter and $\lceil lgn \rceil$ bits to store the current value, but the total is still $O(\log n)$.

2. For this part, the intuition is that we will sample a value as we go, then record a winner if we happen to see it again. How easy this is to analyze depends on which sampling algorithm we use:

   (a) If we pick a particular position i and then sample x_i, we win as long as x_i is not the last occurrence of this particular value in the sequence. Since there are at most n positions that are the last occurrence of a value, we have at least a $1 - n/2n = 1/2$ chance of winning. This requires an extra bit to record whether or not the sampled value appeared again, for a total of $\lceil \lg 3n \rceil + 1 = O(\log n)$ bits.

   (b) Alternatively, we could run the continuous sampling algorithm above and fix the stored value if we see it again. The analysis in this case is a bit more messy. Here is one possible approach.

   Consider the original sampling process, and let A_i be the event that (a) there exists some $j > i$ such that $x_j = x_i$, and (b) value x_i is stored at time i and never overwritten. There are at least n positions for which (a) holds, and for each of these positions we have $\Pr[A_i] = 1/2n$.

   Unfortunately these events are not independent. But for any i, we have that $\Pr[\neg A_i \mid \neg A_1 \wedge \ldots \wedge \neg A_{i-1}]$ is either 1 (if A_i never occurs) or at most $1 - 1/2n$ (since previous events A_j not occurring for $j < i$ i can only increase the likelihood of A_i occurring). The latter case holds for at least n choices of i, giving

$$\begin{aligned}\Pr\left[\bigwedge \neg A_i\right] &= \prod \Pr[\neg A_i \mid \neg A_1 \wedge \ldots \wedge \neg A_{i-1}] \\ &\leq (1 - 1/2n)^n \\ &\leq e^{-1/2},\end{aligned}$$

[1] This is a special case of a more general algorithm known as **reservoir sampling**, due to Vitter [Vit85].

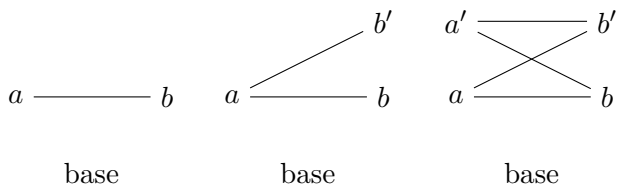


Figure B.2: Example of graph density evolution. Initial graph has density $D_0 = 1/\binom{3}{2} = 1/3$. After duplicating b, new density is $D_1 = 2/\binom{4}{2} = 1/3$. After duplicating a as well, final density is $D_2 = 4/\binom{5}{2} = 2/5$.

which is a constant strictly less than 1. So with constant probability, at least one event A_i occurs. But then the algorithm observes a duplicate and returns it.

B.2.2 A dense network

Suppose we construct a network as follows. Initially, we have $n_0 = n$ nodes, of which one is an inert base station with no incident edges. We also have $m_0 = m$ edges between the $n - 1$ remaining nodes. At each step, we pick one of the $n_t - 1$ non-base-station nodes uniformly at random and duplicate it, creating a new node connected to the same nodes as the old one.

The **density** D_t of this network after t steps is the ratio between the number of edges m_t and the number of possible edges $\binom{n_t}{2}$, where $n_t = n_0 + t$ is the number of nodes (including the base station, even though it doesn't otherwise participate in the network). We'd like to examine how the density evolves over time as new nodes are added to the network.

An example of this process is given in Figure B.2.

1. Compute the exact value of $\mathrm{E}[D_t]$ as a function of n, m, and t.
2. Show that, for any t, D_t is within $O\left(\sqrt{\log n / n}\right)$ of $\mathrm{E}[D_t]$ with high probability.

Solution

1. We'll show that $\{D_t\}$ is a martingale, giving $\mathrm{E}[D_t] = \mathrm{E}[D_0] = m/\binom{n}{2}$ for all t.

   Let G_t be the graph after t steps. Let $\mathcal{F}_t$ be the σ-algebra generated by $\langle G_0, \ldots, G_t \rangle$.

Each edge in G_t goes between two non-base-station nodes, so there is a chance of exactly $\frac{2}{n_t-1}$ that one of its endpoints is duplicated. Summing over all edges gives $\mathrm{E}\left[m_{t+1} \mid m_t\right] = m_t\left(1 - \frac{2}{n-1}\right)$. But then

$$\begin{aligned}
\mathrm{E}\left[D_{t+1} \mid \mathcal{F}_t\right] &= \frac{m_t(2)n - 1}{\binom{n_t+1}{2}} \\
&= \frac{m_t \frac{n+1}{n-1}}{\binom{n_t+1}{2}\frac{n+1}{n-1}} \\
&= \frac{m_t}{\binom{n_t}{2}} \\
&= D_t.
\end{aligned}$$

It follows that $\mathrm{E}\left[D_t\right] = \mathrm{E}\left[D_0\right] = m/\binom{n}{2}$.

2. We will apply Azuma's inequality. This requires getting bounds on the martingale differences $X_t = D_t - D_{t-1}$.

   Since the maximum possible degree of any node in G_{t-1} is $n_{t-1} - 2$, we have $m_t \leq m_{t-1} + n_{t-1} - 2$ and thus

$$\begin{aligned}
D_t - D_{t-1} &\leq \frac{m_{t-1} + n_{t-1} - 2}{\binom{n_{t-1}+1}{2}} - \frac{m_{t-1}}{\binom{n_{t-1}}{2}} \\
&\leq \frac{m_{t-1} + n_{t-1} - 2}{\binom{n_{t-1}}{2}} - \frac{m_{t-1}}{\binom{n_{t-1}}{2}} \\
&\leq \frac{n_{t-1}}{\binom{n_{t-1}}{2}} \\
&\leq \frac{2}{n_{t-1} - 1}. \\
&= \frac{2}{n + t - 2}.
\end{aligned}$$

   In the other direction, duplicating a node destroys no edges, so $m_t \geq$

m_{t-1}, giving

$$\begin{aligned} D_t - D_{t-1} &\geq \frac{m_{t-1}}{\binom{n_{t-1}+1}{2}} - \frac{m_{t-1}}{\binom{n_{t-1}}{2}} \\ &= m_{t-1}\frac{\binom{n_{t-1}}{2} - \binom{n_{t-1}+1}{2}}{\binom{n_{t-1}+1}{2}\binom{n_{t-1}}{2}} \\ &= m_{t-1}\frac{-n_{t-1}}{\binom{n_{t-1}+1}{2}\binom{n_{t-1}}{2}} \\ &\leq \frac{-n_{t-1}}{\binom{n_{t-1}+1}{2}} \\ &= -\frac{2}{n+t}. \end{aligned}$$

So in either direction we have $|X_t| = |D_{t+1} - D_t| \leq \frac{2}{n+t-2}$.

Now let us apply Azuma's inequality:

$$\begin{aligned} \Pr\left[|D_t - \mathrm{E}\left[D_t\right]| \geq \alpha\right] &\leq 2\exp\left(-\alpha^2 / \sum_{i=1}^{t}\left(\frac{2}{n+i-2}\right)^2\right) \\ &\leq 2\exp\left(-\alpha^2 / \left(4\sum_{k=n-1}^{\infty}\frac{1}{k^2}\right)\right) \\ &\leq 2\exp\Big(-\alpha^2 / \Big(4\int_{x=n-2}^{\infty}\frac{1}{x^2}\,dx\Big)\Big) \\ &\leq 2\exp\Big(-\alpha^2(n-2)/4\Big). \end{aligned}$$

For any fixed c, setting $\alpha = \sqrt{4(c+1)\ln n/(n-2)} = O\Big(\sqrt{\log n/n}\Big)$ gives a probability of at most $2n^{-c-1} \leq n^{-c}$ (when $n \geq 2$) that we are more than α away from the expectation.

B.3 Assignment 3, due Thursday 2023-04-27 at 23:59

B.3.1 Shuffling a graph

Consider the following shuffling algorithm: Given a connected graph G with n vertices labeled 1 through n, and m edges, at each step, with probability 1/2, choose one of the edges uniformly at random and swap the labels of

its endpoints. This process yields a Markov chain that is aperiodic and irreducible,[2] so it has a unique stationary distribution.

1. What is the stationary distribution of this process?

2. Show that this process converges to within a total variation distance of ϵ of its stationary distribution in time polynomial in n and $\log \frac{1}{\epsilon}$.

Solution

1. Suppose there is a labeling i and a labeling j such that j is reachable from i in one step. Then these labelings differ by swapping one edge so $p_{ij} = p_{ji} = \frac{1}{2m}$. For each labeling i, let $\pi_i = 1/n!$. Then $\sum_i \pi_i = 1$ and, for all i and j, $\pi_i p_{ij} = \pi_j p_{ji} = \frac{m}{n!}$. This shows that the chain is reversible and that this particular π is its stationary distribution.

2. We'll do a coupling argument, similar to the argument for adjacent-swap shuffling on a line. For technical reasons that we will see later, it will be helpful to assume that G has at least two edges. In the case where G has only one edge, we can argue that the process reaches the stationary distribution in one step, as there are only two labelings and the first step has a 1/2 chance each of switching or staying put.

   Let X and Y be two copies of the process, with X starting in an arbitrary distribution and Y starting in the uniform stationary distribution. At each step, let D be the set of edges with the same label on at least one endpoint. For each edge uv:

   (a) If $uv \in D$, swap uv in both copies with probability $\frac{1}{2m}$.
   (b) If $uv \notin D$, swap uv in X only with probability $\frac{1}{2m}$ and in Y only with probability $\frac{1}{2m}$.

   Do nothing with the remaining probability $\frac{|D|}{2m}$.

   For each label ℓ, write X_ℓ for the vertex labeled with ℓ in X and Y_ℓ for the vertex labeled with ℓ in Y. As in the adjacent-swap case, we can argue that once $X_\ell = Y_\ell$, any edge uv with an endpoint labeled ℓ is in D, so X_ℓ continues to equal Y_ℓ in all subsequent states of the coupled

[2] Aperiodicity is immediate from having steps that do nothing. For irreducibility, if G is a tree, there is at least one vertex v with degree 1; use a sequence of swaps to move the desired label to v, then recurse on $G \setminus \{v\}$ to fix the rest of the labeling. For general G, apply this procedure to a spanning tree of G.

process. So the question then is how long it takes for two unlinked copies of a label to become linked.

Define a new process $Z^t = \langle U^t, V^t \rangle$ where U^t represents the position X_ℓ^t and V^t represents Y_ℓ^t. Unlike the $\langle X_\ell^t, Y_\ell^t \rangle$ process, we will assume that at each step of Z^t we always move U^t across an incident edge with probability $\frac{1}{2n}$ and similarly for V^t, without regard to whether $U^t = V^t$. This gives a Markov chain with the same transition probabilities as $\langle X_\ell^t, Y_\ell^t \rangle$ when $X_\ell^t \neq Y_\ell^t$, but unlike the $\langle X_\ell^t, Y_\ell^t \rangle$ process, the Z^t process is both irreducible and reversible, with a uniform stationary distribution. We can also argue that it is aperiodic under our assumption that $m > 1$, because there exists at least one state where U^t is not adjacent to some edge, so there is a nonzero probability that $Z_{t+1} = Z^t$ in this state.

We will now show that X_ℓ^t and Y_ℓ^t collide with high probability after polynomially-many steps by showing convergence of Z^t to its stationary distribution using Cheeger's inequality. Observe that the Z chain has exactly m^2 states, and that each non-empty subset S of the chain has at least one transition that leaves it with probability $\pi_i p_{ij} = \frac{1}{m^2} \cdot \frac{1}{2m} = \frac{1}{2m^3}$. This gives $\Phi_S \geq \frac{1/2m^3}{|S|/m^2}$ which is at least $\frac{1}{4m^3}$ in the worst case. It follows that $\tau_2 \leq \frac{2}{\Phi^2} \leq 8m^6$ and that the total variation distance between Z^t and the uniform distribution is at most $\frac{1}{2m}$ after $O(m^6 \log m)$ steps. In particular this gives a probability of at least $\frac{1}{m} - \frac{1}{2m} = \frac{1}{2m}$ of being in a state $\langle U^t, V^t \rangle$ with $U^t = V^t$ at the end of each such interval. Repeating for $O(m \log m)$ intervals with a suitable constant gives a probability of at most m^{-c} for any fixed c that $U^t = V^t$ at the end of one of these intervals, which shows that X_ℓ and Y_ℓ collide within $O(m^7 \log^2 m)$ steps with probability at least $1 - m^{-c}$.

Since $m \geq n - 1$, we can choose c so that the probability that X_ℓ and Y_ℓ don't collide in $O(m^7 \log^2 m)$ steps is $O(n^{-2})$. By the union bound, this gives that X_ℓ and Y_ℓ collide in time $O(m^7 \log^2 m)$ with probability at least $1 - O(n^-1)$. Repeat as needed at most $O(\log(1/\epsilon))$ times to drive the probability of failure down below ϵ, and use $m = O(n^2)$ to express the convergence time in terms of n as $O(n^1 4 \log^2 n \log(1/\epsilon))$. This polynomial in n and $\log \frac{1}{\epsilon}$ even though the exponents are terrible.

B.3.2 Counting unbalanced sets

Suppose you are given the usual set balancing set-up, where you have a collection of sets $S_1, S_2, \ldots, S_m$ with $|\bigcup S_i| = n$, and you are asked assign a

value ± 1 to each element x of of $S = \bigcup S_i$ with the goal of minimizing the discrepancy $d = \max_i \left| \sum_{x \in S_i} x \right|$.

We know that applying a random assignment gives $d = O(\sqrt{n \log m})$ with high probability. But this bound doesn't take into account any structure the S_i might have. Give a FPRAS that takes the S_i as input along with a bound t and computes an approximation for $\Pr[d \geq t]$, assuming each x is assigned ± 1 using an independent fair coin-flip.

Solution

Let A be the set of all assignments that produce $d \geq t$. For each i, let $X_i = \sum_{x \in S_i} x$ be the random variable representing the sum of the assignments of elements of S_i. We can write A as the union of B_{ij} for all $i \in [m]$ and all j with $|j| \geq t$, where B_{ij} is the event that $X_i = j$. Since this union is not disjoint, we will use Karp-Luby (see §11.4) to approximate its size.

As with #DNF, the idea is that we can both count and sample triples $\langle i, j, x \rangle$ where x is an assignment that makes $X_i = j$. Let $n_i = |S_i|$. Let Y_i be the number of elements of S_i that are assigned $+1$; then $X_i = Y_i - (n_i - Y_i) = 2Y_i - n_i$, and solving for Y_i gives $Y_i = \frac{n_i + X_i}{2}$, assuming the numerator is even. Given i and j, there are exactly $2^{n-n_i} \binom{n_i}{(n_i+j)/2}$ assignments with $X_i = j$ when $n_i + j$ is even, and none when $n_i + j$ is odd. For the even case, we can sample these assignments in polynomial time by choosing $(n_i + j)/2$ elements without replacement from S_i to have value $+1$, making the remaining elements of S_i have value -1, and assigning ± 1 values independently to any $x \in S \setminus S_i$. We will abuse notation slightly by writing B_{ij} for the set of all possible triples $\langle i, j, x \rangle$ resulting from this process for a particular pair of values i and j. Let $B = \bigcup_{i=1}^{n} \bigcup_{|j| \geq d, j + n_i \text{ even}} B_{ij}$; note that this union is disjoint.

Our full sampling procedure looks like this:

1. Sample i and j with probability $|B_{ij}|/|B|$. Assuming that we precompute the sizes of the B_{ij} sets and that arithmetic operations take time $O(1)$ time each, this takes time at most $O(nm)$ if we are naive about the sampling and $O(\log nm)$ if we are a little bit more clever and use binary search.

2. Sample an element $\langle i, j, x \rangle$ of B_ij uniformly as described above. This takes time $O(n)$.

3. Return 1 if x does not appear in $B_{i'j'}$ for any $i'j'$ that precedes ij in lexicographic order. This requires checking $O(nm)$ previous sets at a

cost of $O(n)$ each, for $O(n^2m)$ time total, which dominates the cost of generating the sample.

Let ρ be the probability that we obtain 1 in an iteration of this process. Then $|A| = \rho|B|$, and we can approximate $|A|$ to within relative error ϵ by approximating ρ within the same bound. Because each element of A occurs in at least one and at most $O(nm)$ of the B_{ij}, we have that $\rho = \Omega\left(\frac{1}{nm}\right)$. Applying Lemma 11.2.1, approximating ρ using N samples where $N = \frac{3}{\epsilon^2\rho}\ln\frac{2}{\delta} = O\left(\frac{n^2m^2\log\delta}{\epsilon^2}\right)$ gives us our desired error bounds in time polynomial in nm, $1/\epsilon$, and $\log\delta$, as required.

Appendix C

Sample assignments from Fall 2019

C.1 Assignment 1: due Thursday, 2019-09-12, at 23:00

Bureaucratic part

Send me email! My address is james.aspnes@gmail.com.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Whether you are taking the course as CPSC 469 or CPSC 569.
4. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

C.1.1 The golden ticket

An infamous candy company announces it is placing a golden ticket uniformly at random in one of n candy bars that are distributed to n children, with unimaginable wealth to be bestowed on the child who receives the ticket.

Unfortunately, the candy company owner is notorious for trickery and lies, so the probability that the golden ticket actually exists is only 1/2.

Consider the last child to open their candy bar, and what estimate they should make of their probability of winning after seeing the first k children open their candy bars to find no ticket.

An optimist reasons: If the ticket does exist, then the last child's chances went from $1/n$ to $1/(n-k)$. So the fact that none of the first k children got the ticket is good news!

A pessimist reasons: The more candy bars are opened without seeing the ticket, the more likely it is that the ticket doesn't exist. So the fact that none of the first k children got the ticket is bad news!

Which of them is right? Compute the probability that the last child receives the ticket given the first k candy bars come up empty, and compare this to the probability that the last child receives the ticket given no information other than the initial setup of the problem.

Solution

Let W be the event that the last child wins, C the event that the candy bar exists, and L_k the event that the first k children lose.

Without conditioning, we have

$$\begin{aligned}\Pr[W] &= \Pr[W \mid C]\Pr[C] + \Pr[W \mid \neg C]\Pr[\neg C] \\ &= \frac{1}{n}\cdot\frac{1}{2} + 0\cdot\frac{1}{2} \\ &= \frac{1}{2n}.\end{aligned}$$

With conditioning, we have

$$\begin{aligned}
\Pr[W \mid L_k] &= \frac{\Pr[W \wedge L_k]}{\Pr[L_k]} \\
&= \frac{1/2n}{\Pr[L_k]\, C \Pr[C] + \Pr[L_k]\, \neg C \Pr[\neg C]} \\
&= \frac{1/2n}{\frac{n-k}{n} \cdot \frac{1}{2} + 1 \cdot \frac{1}{2}} \\
&= \frac{1/2n}{\frac{n-k}{2n} + \frac{1}{2}} \\
&= \frac{1/2n}{\frac{2n-k}{2n}} \\
&= \frac{1}{2n-k}.
\end{aligned}$$

Since $\frac{1}{2n-k} > \frac{1}{2n}$ when $k > 0$, the optimist is correct.

C.1.2 Exploding computers

Suppose we have a system of $3n$ machines $m_0, m_2, \ldots m_{3n-1}$ in a ring, and we turn each machine on at random with independent probability p. Over time, the machines may overheat: if we turn on three consecutive machines m_i, m_{i+1}, and m_{i+2} (wrapping indices around mod $3n$), the middle machine m_{i+1} overheats and explodes. Explosions happen simultaneously and are purely a function of which machines are initially turned on; so if m_1, m_2, m_3, m_4, and m_5 are all turned on, m_2, m_3, and m_4 will all explode.

Call a machine a **active** if it is turned on and does not explode.

We want to choose p to maximize the number of active machines X. But there are two ways to interpret this goal.

1. Suppose our goal is to maximize the *expected* number of active machines $\mathrm{E}[X]$. What value of p should we choose, and what does this give us for $\mathrm{E}[X]$?

2. Suppose instead our goal is to maximize the probability of having the maximum possible number of active machines. More formally, letting M be the number of survivors we could get if we chose which machines to turn on optimally, we want to choose p to maximize $\Pr[X = M]$. What value of p should we choose, and what does this give us for $\Pr[X = M]$?

Solution

1. Let A_i be the event that machine m_i is active, and let X_i be the indicator variable for m_i being active and not exploding. We are trying to maximize $\mathrm{E}\left[\sum X_i\right] = \sum \mathrm{E}\left[X_i\right] = n\,\mathrm{E}\left[X_1\right]$, where the last equation holds by symmetry.

   Observe that $X_1 = 1$ if A_1 occurs and it is not the case that both A_0 and A_2 occur. This gives $\mathrm{E}\left[X_1\right] = \Pr\left[X_1 = 1\right] = p(1-p^2) = p - p^3$. By setting the derivative $1 - 3p^2$ to zero we find a unique extremum at $p = \sqrt{1/3}$; this is a maximum since $p(1-p^2) = 0$ at the two corner solutions $p = 0$ and $p = 1$. Setting $p = \sqrt{1/3}$ gives $3n\,\mathrm{E}\left[X_i\right] = 3n \cdot \sqrt{1/3}(2/3) = n\frac{2}{\sqrt{3}} \approx n \cdot 1.154701\ldots$.

2. First we need to figure out M.

   We can assume that we set up the optimal configuration so that none explode, because any initial configuration that produces explosions can be replaced by one where the dead machines are just turned off to start with, with no reduction in the number of active machines.

   Conveniently, we can divide the machines into consecutive groups of three, and observe that (a) turning on the first two machines in each group causes no explosions, (b) turning on three machines in any group causes the middle one to explode. This shows $M = 2n$ is the maximum achievable number of machines, because (a) lets us get M and (b) says we can't do more, since this would give us a group with three by the Pigeonhole Principle.

   We still need to characterize how many possible patterns of $M = 2n$ machines produce no explosions. Since we can only have two consecutive active machines, each pair of active machines must be followed by an inactive machine. Putting two inactive machines in a row would allow a partitioning into groups of three with some group getting only one active machine, leaving some other group with three, which is trouble. So the pattern must consist of pairs of active machines alternating with single inactive machines. There are exactly three ways to do this.

   Each of these three patterns occurs with probability $p^{2n}q^n$ where $q = 1-p$. Maximizing this probability can again be done by taking the derivative $2np^{2n-1}q^n - np^{2n}q^{n-1}$ and setting to 0. Removing common factors gives $2q - p = 0$, or $2 - 3p = 0$, giving $p = 2/3$, which is about what one would expect.

We then get $\Pr[X = 2n] = 3 \cdot (2/3)^{2n}(1/3)^n$. This is pretty small for reasonably large n, which suggests that a randomized algorithm may not be the best in this case.

C.2 Assignment 2: due Thursday, 2019-09-26, at 23:00

C.2.1 A logging problem

An operating system generates a new log entry at time t with independent probability p_t, and the size of each log entry for any t is an integer X chosen uniformly in the range $a \le X \le b$, where the choice of X at a particular time t is independent of all the other log entry sizes and all choices of whether to write a log entry.

Suppose that over some fixed period of time, the operating system generates n log entries on average. Compute the expectation of the total size S of all the log entries, and show that the probability that S is more than a small constant fraction larger than $\mathrm{E}[S]$ is exponentially small as a function of n.

Solution

For each t, let X_t be the size of the log entry at time t. Then $\mathrm{E}[S] = \sum_t \mathrm{E}[X_t] = \sum_t p_t \frac{a+b}{2} = \frac{a+b}{2} \mathrm{E}[\sum_t p_t] = \frac{a+b}{2} \cdot n$.

We can't use Chernoff bounds directly on the X_t because they aren't Bernoulli random variables, and we can't use Hoeffding's inequality because we don't actually know how many of them there are. So we will use Chernoff first to bound the number of nonzero X_t and then use Hoeffding to bound their sum.

The easiest way to think about this is to imagine we generate the nonzero X_t by generating a sequence of independent values $Y_1, Y_2, \ldots$, with each Y_i uniform in $[a, b]$, then select the i-th nonzero X_t to have value Y_i.

Let Z_t be the indicator variable for the event that a log entry is written at time t. Then $N = \sum_t Z_t$ gives the total number of log entries, and $\mathrm{E}[N] = \mathrm{E}[\sum_t Z_t] = \sum_t p_t = n$.

Pick some $\delta < 1$; then Chernoff's inequality (5.2.2) gives

$$\Pr[N \ge (1+\delta)n] \le e^{-n\delta^2/3}.$$

Suppose $N < (1+\delta)n$. Then $\sum_t X_t = \sum_{i=1}^{N} Y_i \le \sum_{i=1}^{(1+\delta)n} Y_i$. This last quantity is a sum of independent bounded random variables, so we can apply

the asymmetric version of Hoeffding's inequality (5.3.17) to get

$$\Pr\left[\sum_{i=1}^{(1+\delta)n} Y_i - (1+\delta)n \cdot \frac{a+b}{2} \geq s\right] \leq \exp\left(-\frac{s^2}{2(1+\delta)n\left(\frac{b-a}{2}\right)^2}\right)$$

For $s = \delta n \cdot \frac{a+b}{2}$ this gives a bound of

$$\begin{aligned}\exp\left(-\frac{-\delta^2 n^2\left(\frac{a+b}{2}\right)^2}{2(1+\delta)n\left(\frac{b-a}{2}\right)^2}\right) &\leq \exp\left(-\frac{-\delta^2 n\left(\frac{a+b}{2}\right)^2}{2(1+\delta)\left(\frac{b+a}{2}\right)^2}\right) \\ &= \exp\left(-\frac{-\delta^2 n}{2(1+\delta)}\right).\end{aligned}$$

If we put this all together, we get

$$\Pr\left[S \geq (1+2\delta)n\frac{a+b}{2}\right] \leq e^{-n\delta^2/3} + e^{-n\delta^2/2(1+\delta)} = e^{-\Theta(n)}.$$

C.2.2 Return of the exploding computers

Suppose we are working in the model described in §C.1.2, where $3n$ machines in a ring are each turned on independently at random with probability p, and a machine remains active only if it is turned on and at least one of its neighbors is not turned on. We have seen that an optimal choice of p gives cn active machines on average for a constant c independent of n.

Unfortunately, averages are not good enough for our customers. Show that there is a choice of p and constants $c' > 0$ and $\delta > 0$ such that at least $c'n$ machines are active with probability at least $1 - e^{-\delta n}$ for any n.

Solution

We can use the method of bounded differences. Let X_i be the indicator for the event that machine i is turned on, and let $S = f(X_1, \ldots, X_{3n})$ compute the number of active machines. Then changing one input to f from 0 to 1 increases f by at most 1, and decreases it by at most 2 (since it may be that the new machine explodes, producing no increase, and it also explodes both neighbors, producing a net -2 change). So McDiarmid's inequality (5.3.13) applies with $c_i = 2$ for all i, giving

$$\Pr\left[f(X_1, \ldots, X_{3n} - \mathrm{E}\left[f(X_1, \ldots, X_{3n}\right] \leq t\right] \leq \exp\left(-\frac{2t^2}{3n \cdot 2^2}\right) \quad = e^{-t^2/6n}.$$

Choose p to maximize the expected number of active machines, and let $\mathrm{E}[S] = c'n$. Choose c so that $0 < c < c'$. Then $\Pr[S \le cn] = \Pr[S - \mathrm{E}[S] \le c'n - cn] \le e^{-(c'-c)^2n^2/6n} = e^{-(c'-c)^2n/6} = e^{-\delta n}$, where $\delta = (c'-c)^2/6 > 0$.

The above is my original sample solution, below are some quick sketches of alternative approaches that showed up in submissions:

- Analyze the dependence and use Lemma 5.2.2. This gives similar results to McDiarmid's inequality with a bit more work. The basic idea is that $\mathrm{E}[X_i | X_1 \dots X_{i-1}] \ge p(1-p)$ for $i < 3n$ whether or not X_{i-1} is 1 (we don't care about the other variables). So we can let $p_i = p(1-p)$ for these values of i. To avoid wrapping around at the end, either omit X_{3n} entirely or let $p_{3n} = 0$.

- Even simpler: Let Y_i be the indicator for the event that machine i is active. Then $Y_3, Y_6, \dots, Y_{3n}$ are all independent, and we can bound $\sum_{i=1}^{n} Y_{3i} \le \sum_{i=1}^{3n} Y_i$ using either Chernoff or Hoeffding.

C.3 Assignment 3: due Thursday, 2019-10-10, at 23:00

C.3.1 Two data plans

A cellphone company offers a choice of two data plans, both based on penalizing the user whenever they use more data than in the past. For both plans the assumption is that your data usage in each month $i \in \{0 \dots \infty\}$ is an independent continuous random variable that is uniform in the range $[0, c]$ for some fixed constant c.

With **Plan A**, you pay a penalty of \$1 for every month i in which your usage X_i exceeds $\max_{j<i} X_j$. This excludes month 0, for which the max of the previous empty set is taken to be $+\infty$.

With **Plan B**, you pay a penalty of \$100 for every pair of months $2i$ and $2i+1$ in which X_{2i} and X_{2i+1} both exceed $\max_{j<2i} X_j$. As in Plan A, you never pay a penalty for the pair of months 0 and 1.

1. Suppose that you expect to live forever (and keep whatever plan you pick forever as well). If your goal is to minimize your expected total cost over eternity, which plan should you pick?

2. Which plan is cheaper on average if you don't plan to live for more than, say, the next 1000 years?

(Assume no discounting of future payments in either case.)

Solution

Let A_i be the event that we pay a penalty in month i with plan A, and let B_i for even i be the event that we pay a penalty for months i and $i+1$ in plan B. Then for $i > 0$,

$$\begin{aligned}\Pr[A_i] &= \Pr\left[X_i > \max_{j<i} X_j\right] \\ &= \Pr\left[X_i = \max_{j\le i} X_i\right] \\ &= \frac{i!}{(i+1)!} \\ &= \frac{1}{i+1}.\end{aligned}$$

Similarly,

$$\begin{aligned}\Pr[B_i] &= \Pr\left[X_i > \max_{j<i} X_j \text{ and } X_{i+1} > \max_{j<i}\right] \\ &= \frac{i! \cdot 2}{(i+2)!} \\ &= \frac{2}{(i+2)(i+1)}.\end{aligned}$$

(In the first case, we count all orderings of the $i+1$ elements $0 \ldots i$ that make X_i largest, and divide by the total number of orderings. In the second case, we count all the orderings that make X_i and X_{i+1} both larger than all the other values.)

Now let us do some sums.

1. For the infinite-lifetime version, Plan A costs

$$\sum_{i=1}^{\infty} \frac{1}{i+1},$$

which diverges. Plan B costs

$$\begin{aligned} 100 \cdot \sum_{i=1}^{\infty} \frac{2}{(2i)(2i-1)} &\leq 100 \cdot \sum_{i=1}^{\infty} \frac{2}{(2i)^2} \\ &= 100 \cdot \sum_{i=1}^{\infty} \frac{1}{2i^2} \\ &= \frac{100}{2} \cdot \frac{\pi^2}{6}. \end{aligned}$$

which is finite. So Plan B is better for immortals.

2. For the finite-lifetime version, over n months, Plan A costs

$$\sum_{i=1}^{n} \frac{1}{i} \approx \ln n + \gamma$$

Plan B has a $\frac{2}{3 \cdot 4} = \frac{1}{6}$ chance of costing \$100 after the first four months, giving an expected cost of at least \$16. As long as you keep both for at least four months, for $n \ll e^{16} \approx 8886110$ months ≈ 740509 years, plan A is better.

C.3.2 A randomly-indexed list

Consider the following data structure: We store elements in a sorted linked list, and as each element is inserted, with probability p we also insert it in a balanced binary search tree data structure with guaranteed $O(\log n)$-time search and insert operations (say, a red-black tree). We can now search for an element x by searching for the largest element y in the tree less than or equal to x, then walking along the list starting from y until we find x (or some value bigger than x).

Recall that for a skip list with (small) parameter p, the expected number of pointers stored per element is $1 + O(p)$ and the expected cost of a search operation is $O\left(\frac{1}{p} \log n\right)$.

1. What are the corresponding asymptotic values for the expected pointers per element and the expected cost of a search for this new data structure, as a function of p and n?

2. Suppose you are given n in advance. Is it possible to choose p based on n for the tree-based data structure to get lower expected space overhead (expressed in terms of pointers per element) than with a skip list, while still getting $O(\log n)$ expected search time?

Solution

1. This is actually easier than the analysis for a skip list, because there is no recursion to deal with.

   A balanced binary search tree stores exactly 2 pointers per element, so with an expected pn elements in the tree, we get $n + 2pn$ pointers total, or $1 + 2p$) pointers per element.

   For the search cost, work backwards from the target x to argue that we traverse an expected $O(1/p)$ edges in the linked list before hitting a tree node. Searching the tree takes $O(\log n)$ time independent of p. This gives a cost of $O\left(\frac{1}{p} + \log n\right)$.

2. Yes. Let $p = \Theta(1/\log n)$. Then we use an expected $1 + O(1/\log n)$ pointers per element, while the search cost is $O\left(\frac{1}{1/\log n} + \log n\right) = O(\log n)$. In contrast, getting $1 + O(1/\log n)$ expected space with a skip list would also require setting $p = \Theta(1/\log n)$, but this would give an expected search cost $\Theta(\log^2 n)$.

C.4 Assignment 4: due Thursday, 2019-10-31, at 23:00

C.4.1 A hash tree

Suppose we attempt to balance a binary search tree by inserting n elements one at a time, using hashes of the values instead of the values themselves.

Specifically, let $M = \{1 \ldots n^2\}$ be the set of possible hash values; let $U = \{1 \ldots u\}$ be the universe of possible values, where $u \gg n^2$; and let H be a 2-universal family of hash functions $h : U \to M$. After the adversary chooses n distinct values $x_1, \ldots, x_n$, the algorithm chooses a hash function $h \in H$ uniformly at random, and inserts x_1 through x_n in order into a binary search tree with no rebalancing, where the key for each value x_i is $h(x_i)$.

In the case of duplicate keys (which shouldn't happen very often), assume that if we try to insert x into a subtree with root r, and $h(x) = h(r)$, we just adopt some fixed rule like always inserting x into the right subtree of r.

For any n and $u \gg n^2$, prove or disprove: for any 2-universal family H of hash functions, and any sequence of values $x_1, \ldots, x_n$, the expected depth of a tree constructed by the above process is $O(\log n)$.

Solution

Disproof: We will construct a 2-universal family H such that $h(x) = x$ for all $h \in H$ whenever $1 \le x \le m$. We can then insert the values $x_i = i$ with corresponding hash values $h(x_i) = i$ and get a tree of depth exactly n.

Start with a *strongly* 2-universal family H'. For each h' in H, construct a corresponding h according to the rule $h(x) = x$ when $1 \le x \le m$ and $h(x) = h'(x)$ otherwise. We claim that this gives a 2-universal family of hash functions.

Recall that H is 2-universal if, for any $x \ne y$, $\Pr[h(x) = h(y)] \le 1/m$, when h is chosen uniformly from H. We consider three cases:

1. If $1 \le x \le m$ and $1 \le y \le m$, then $h(x) = x \ne y = h(y)$ always.

2. If $1 \le x \le m$ and $m < y$, then $\Pr[h(y) = h(x)] = \Pr[h'(y) = x] = 1/m$, since $h(y)$ is equally likely to be any value by strong 2-universality. A symmetric argument works if $m < x$ and $1 \le y \le m$.

3. If $m < x$ and $m < y$, then $\Pr[h(x) = h(y)] = \Pr[h'(x) = h'(y)] = 1/m$.

In each case we have $\Pr[h(x) = h(y)] \le 1/m$. So H gives a 2-universal hash family that produces trees of depth n, much greater than $O(\log n)$.

C.4.2 Randomized robot rendezvous on a ring

Suppose we have a ring of size nm, consisting of positions $0, \dots, nm - 1$, with n robots starting at positions $X_{i0} = im$.

At each time step t, an adversary chooses one of the n robots i, and this robot moves from position X_{it} to $X_{i,t+1} = (X_{it} \pm 1) \bmod mn$, moving clockwise or counterclockwise with equal probability. If any other robot or robots i' are at the same position $X_{i't} = X_{it}$, they also move to the same new position with robot i. Eventually, the robots slowly coalesces until all robots are in the same position.

The process finishes at the first time τ where $X_{i\tau} = X_{j\tau}$ for all i and j. What is $\mathrm{E}[\tau]$?

(Assume that the adversary's choice of robot to move at each time t doesn't involve predicting the future. Formally, this choice must be measurable $\mathcal{F}_t$, where $\mathcal{F}_t$ is the σ-algebra generated by all the positions of the robots at times up to and including t.)

Solution

We'll use a variant of the $X_t^2 - t$ martingale for random walks.

For each i and t, let $Y_{it} = X_{i+1,t} - X_{it}$ be the size of the gap between X_{i+1} and X_i (wrapping around in the obvious way).

Suppose at some step t the adversary chooses robot j to move. Let i and k be the smallest and largest robot ids such that $X_{it} = X_{jt} = X_{kt}$ (again wrapping around in the obvious way).

Then $Y_{i-1,t}$ and Y_{kt} are the only gaps that change, and each rises or drops by 1 with equal probability. So $\mathrm{E}\left[Y_{i-1,t+1}^2 \;\middle|\; \mathcal{F}_t\right] = Y_{i-1,t}^2 + 1$ and $\mathrm{E}\left[Y_{j,t+1}^2 \;\middle|\; \mathcal{F}_t\right] = Y_{j,t}^2 + 1$. If we define $Z_t = \sum_{i=1}^n Y_{it}^2 - 2t$, then $\{Z_t\}$ is a martingale with respect to $\{\mathcal{F}_t\}$.

For any adversary strategy, starting from any configurations, there is a sequence of at most $n^2 m$ coin-flips that causes all robots to coalesce. So $\mathrm{E}[\tau] \le n^2 m 2^{n^2 m} < \infty$. We also have that $|Z_{t+1} - Z_t| \le 2$, so we can apply the finite-expectation/bounded-increments case of Theorem 9.3.1 to show that $\mathrm{E}[Z_\tau] = \mathrm{E}[Z_0] = nm^2$. But at time τ, all but one interval Y_{it} is zero, and the remaining interval has length mn. This gives $\mathrm{E}[Z_\tau] = \mathrm{E}\left[(mn)^2 - 2\tau\right]$, giving $\mathrm{E}[\tau] = \frac{1}{2}(n^2m^2 - nm^2) = \binom{n}{2}m^2$.

The nice thing about this argument is that it instantly shows that the adversary's strategy doesn't affect the expected time until all robots coalesce, but the price is that the argument is somewhat indirect. For specific adversary strategies, it may be possible to show the coalescence time more directly. For example, consider an adversary that always picks robot 0. Then we can break down the resulting process into a sequence of phases delimited by collisions between 0 and robots that have not yet been added to its entourage. The first such collision happens after robot 0 hits one of the two absorbing barriers at $-m$ and $+m$, which occurs in m^2 steps on average. At this point, we now have a new random walk with absorbing barriers at $-m$ and $+2m$ relative to the position of robot 0 (possibly after flipping the directions to get the signs right). This second random walk takes $2m^2$ steps on average to hit one of the barriers. Continuing in this way gives us a sequence of random walks with absorbing barriers at $-m$ and $+km$ for each $k \in \{1 \dots n-1\}$. The total time is thus $\sum_{k=1}^{n-1} km^2 = \frac{(n-1)n}{2}m^2 = \binom{n}{2}m^2$, as shown above for the general case.

C.5 Assignment 5: due Thursday, 2019-11-14, at 23:00

C.5.1 Non-exploding computers

Engineers from the Exploding Computer Company have devised a new scheme for managing the ring of overheating computers from Problem C.1.2. In the new scheme, we start with an n-bit vector $S^0 = 0^n$. Each 1 represents a computer that is turned on, and each 0 represents a computer that is turned off.[1]

Call a state S **permissible** if there is no j such that $S_j^{t+1} = S_{j+1}^{t+1} = S_{j+2}^{t+1} = 1$, where the indices wrap around mod n.

At each step, some position i is selected uniformly at random. If $S_i^t = 1$, S_i^{t+1} is set to 0 with probability p. If $S_i^t = 0$, S_i^{t+1} is set to 1 if this creates a permissible configuration, and stays 0 otherwise. For any positions $j \neq i$, $S_j^{t+1} = S_j^t$.

1. Show that when $0 < p < 1$, the sequence of values S^t forms a Markov chain with states consisting of all permissible sets, and that this chain has a unique stationary distribution π in which $\pi(S) = \pi(T)$ whenever $|S| = |T|$, where $|S|$ is the number of 1 bits in S.

2. Let τ be the first time at which $|S^\tau| \geq n/2$. Show that for any n, there is a choice of $p \geq 0$ such that $\mathrm{E}[\tau] = O(n \log n)$.

Solution

1. First observe that the update rule for generating S^{t+1} depends only on the state S^t; this gives that $\{S^t\}$ is a Markov chain. It is irreducible, because there is always a nonzero-probability path from any permissible configuration S to any permissible configuration T that goes through the empty configuration 0^n. It is aperiodic, because for any nonempty configuration S^t, there is a nonzero probability that $S^{t+1} = S^t$, since we can pick a position i with $S_i^t = 1$ and then choose not set S_i^{t+1} to 0. So a unique stationary distribution π exists.

   We can show that $\pi(S)$ depends only on $|S|$ by using the fact that the Markov chain is reversible. Let S and T be reachable states that differ

[1] The marketing department convinced the company to change the number of computers in the ring from $3n$ to n in response to consumer complaints. If the engineers' scheme works, the next meeting of the marketing department will consider choosing a new name for the company.

only in position i. Suppose that $S_i^t = 0$ and $T_i^t = 1$. Then $p_{ST} = 1/n$ and $p_{TS} = p/n$. Let $c = \sum_S p^{-|S|}$, where S ranges over all admissible states, and let $\pi_S = p^{-|S|}/c$. Pick some S and T as above and let $k = |S|$.

$$\begin{aligned}\pi_S p_{ST} &= \frac{p^{-k}}{c}\frac{1}{n} \\ &= \frac{p^{-k-1}}{c}\frac{p}{n} \\ &= \pi_T p_{TS}.\end{aligned}$$

So this particular choice of π satisfies the detailed balance equations and is the stationary distribution of the chain. Since $\pi(S)$ only depends on $|S|$, it is immediate that $\pi(S) = \pi(T)$ when $|S| = |T|$.

2. For this part, we'll set $p = 0$, then walk directly up to a configuration with $|S^t| \geq n/2$ in $O(n \log n)$ steps on average. Setting $p = 0$ means that the chain is no longer irreducible, but we won't care about this since we are not worried about convergence.

   Let S be a permissible configuration with $|S| = k$. We want to get a lower bound on the number of positions i such that $S[i/1]$ is also permissible. There $n - k$ zeroes in S; each such 0 can be switched to 1 unless it would create three ones in a row. We will show that at most k of the $n - k$ zeroes cannot be switched, by constructing an injection f from the set of unswitchable 0 positions to the set of 1 positions.

   Let i be a position such that $S_i = 0$ but $S[i/1]$ is not permissible. Then one of the following cases holds:

   (a) $S_{i-2} = S_{i-1} = 1$. Let $f(i) = i - 1$.

   (b) $S_{i-1} = S_{i+1} = 1$. Let $f(i) = i - 1$.

   (c) $S_{i+1} = S_{i+2} = 1$. Let $f(i) = i + 1$.

   To show that this is an injection, observe that the only way for $f(i) = f(j)$ to occur is if $f(i) = i + 1$ and $f(j) = j - 1$ (or vice versa). But if $f(i) = i + 1$, then $S_j = S_{i+2} = 1$, and S_j is not an unswitchable zero.

   So out of $n - k$ zeroes we have at most k unswitchable zeroes, giving at least $n - 2k$ switchable zeroes. The expected waiting time to hit one of these at least $n - 2k$ switchable zeroes is at most $\frac{n}{n-2k}$, giving a total expected waiting time of at most $\sum_{k=0}^{\lfloor (n-1)/2 \rfloor} \frac{n}{n-2k} \leq nH_n = O(n \log n)$.

I suspect the correct expected waiting time is in fact $O(n)$, because (a) the pattern 11011 includes at least one 1 that is not the image of any unswitchable zero; (b) after, say, $n/8$ steps it is likely (via linearity of expectation and McDiarmid's inequality) that there are a linear number ℓ of such patterns in S; so that (c) even as k gets close to $n/2$, the number of unswitchable zeros is at least $n - 2k + \ell = \Theta(n)$, giving constant expected waiting times to find the next switchable zero. But actually proving (b) given all the dependencies flying around looks at least mildly painful, so we may want to just hope that an $O(n \log n)$ bound is good enough.

C.5.2 A wordy walk

Consider the following random walk on strings over some alphabet Σ. At each step:

1. With probability p, if $|X^t| > 1$, delete a character from a position chosen uniformly at random from all positions in X^t. If $|X^t| = 1$, do nothing.

2. With probability q, insert a new character chosen uniformly at random from Σ into X^t. The new character will be inserted after the first i characters in X^t, where i is chosen uniformly at random from $\{0 \dots |X^t|\}$.

3. With probability r, choose one of the character positions in X^t uniformly at random and replace the character in that position with a character chosen uniformly at random from Σ.

For example, here is the result of running the above process for a few steps with Σ the lowercase Latin alphabet, $p = 1/2$, $q = 1/4$, and $r = 1/4$, starting from the string `markov`: `markov markzv marzv carzv caurzv cauzv cauav cauav cavav avav`.

Assume $p + q + r = 1$, $p > q > 0$, and $r > 0$.

Suppose that we start with $|X^0| = n$. Show that for any constants p, q, r satisfying the above constraints, and any constant $\epsilon > 0$, there is a time $t = O(n)$ such that $d_{TV}(X^t, \pi) \leq \epsilon$, where π is a stationary distribution of this process.

Solution

We can simplify things a bit by tracking $X^t = |S^t|$. Since changes in the length of S don't depend on the characters in S, $\{X^t\}$ is also a Markov

chain, with transition probabilities

$$p_{ij} = \begin{cases} p & j = i - 1 \\ q & j = i + 1 \\ r & \text{otherwise} \end{cases}$$

Applying the detailed balance equations gives a stationary distribution ρ for X^t of $\rho_i = (q/p)^{i-1}/(1 - q/p)$. By symmetry, this gives a stationary distribution $\pi_S = \frac{(q/p)^{|S|-1}|\Sigma|^{-|S|}}{1-q/p}$. For X with distribution ρ, we have $\mathrm{E}[X]$ finite when $p > q$ since $\mathrm{E}[X] = \frac{1}{1-q/p}\sum_{n=1}^{\infty} n(q/p)^{n-1}$ converges.

We'll show convergence for S^t starting from a particular S^0 using a coupling with a stationary copy of the chain $\{T^t\}$.

Our strategy will go as follows:

1. Let $m = |T^0|$. We will argue that a coupling that applies the same operation (delete, insert, change) to both copies of the chain reaches a configuration where $|S^t| = |T^t| = 1$ in $O\max(n, m)$ expected steps.

2. From a state with $|S^t| = |T^t| = 1$, with probability $r > 0$, both chains switch to a new character. Make this the same character and get $S^t = Y^t$.

3. If this doesn't work, repeat the argument from the start. Note that we now have $|S^t| = |T^t| \leq 2$, so after $O(1)$ expected steps we are back at $|S^t| = |T^t| = 1$ again. The waiting time for convergence is now geometric and has expectation $O(1)$.

Let us now justify the claim that the waiting time to reach $|S^t| = |T^t| = 1$ is $O(\max(n, m))$. This is just the usual argument for a biased random walk with one absorbing barrier. Let $Z^t = \max(|S^t|, |Y^t|)$. Let τ be the stopping time at which Z^t is first equal to 1. Observe that for $t < \tau$, $\mathrm{E}\left[Z^{t+1}\right] Z^0 \dots Z^t = Z^t + q - p$, so $Q^t = Z^t + (p - q)t$ is a martingale with bounded increments, and we can apply OST to get that $\mathrm{E}[Q_\tau] = 1-(p-q)\tau = \mathrm{E}[Q_0] = \max n, m$. This gives $\mathrm{E}[\tau] = \frac{\max n, m-1}{p-q}$.

We now have a coupling that gives an expected d coalescence time of $O(\max(n, m))$. If $\max(n, m) = n$, we are done. But it could be that $\max(n, m) = m$. Conditioning on $|T^t| > n$, we are truncating the distribution of $|T^t|$ to be a geometric distribution starting at $n+1$; this has expected value $n + O(1)$, giving $\mathrm{E}[\max(n, m)] = n + O(1)$ in either case. So the coalescence time is $O(n + O(1)) = O(n)$.

Since ϵ is a constant, after applying the Coupling Lemma and Markov's inequality, the O eats it.

C.6 Assignment 6: due Monday, 2019-12-09, at 23:00

C.6.1 Randomized colorings

Suppose we want to use a constant number k of colors to color the vertices of a graph G with n vertices and m edges so that we minimize the number of monochromatic edges. A particularly simple approach is to color the vertices independently and uniformly at random. This gives m/k monochromatic edges on average. But we'd like to derandomize the algorithm to get at most m/k monochromatic edges always.

There are two standard ways to do this:

1. Replace the independent random colors with pairwise independent random colors and then consider all possible random choices. Show that we can do this by generating n pairwise-independent colors from a small number of random values, and determine the asymptotic time complexity, as a function of n, m, and k, of generating and testing all the resulting colorings to find the best one.

2. Apply the method of conditional probabilities and assign colors one at a time to minimize the expected number of monochromatic edges conditioned on the assignment so far. Show that we can do so in this case, and compute the asymptotic time complexity as a function of n, m, and k of the resulting deterministic algorithm.

Solution

1. The obvious way to generate n pairwise-independent random values in $\{0, \dots, k-1\}$ is to apply the subset-sum construction described in §5.1.2.1. Let $\ell = \lceil \lg(n+1) \rceil$, assign a distinct nonempty subset S_v of $\{1, \dots, \ell\}$ to each vertex v, and let $X_v = \sum_{i \in C_v} r_i$ where $r_1, \dots, r_\ell$ are independent random values chosen uniformly from $\mathbb{Z}_k$. Then the X_v are pairwise independent, giving a probability of $1/k$ that any particular edge is monochromatic. Since this gives m/k monochromatic edges on average, enumerating all choices of $r_1, \dots, r_\ell$ will find a particular coloring that gets at least m/k monochromatic edges.

   There are $k^\ell = k^{\lceil \lg(n+1) \rceil}$ possible choices. This is bounded by $k^{2+\lg n} = O\Big(k^2 n^{\lg k}\Big) = O\Big(n^{\lg k}\Big)$, which is polynomial in n.

   Generating each coloring takes $O(n \log n)$ time, and testing the edges takes $O(m)$ time, for a total time of $O(n^{\lg k}(m + n \log n))$.

With some data structure tinkering we can reduce the complexity a bit. We can enumerate all of $r_1, \ldots, r_k$ with an amortized $O(1)$ values changing at each iteration. This still requires changing about half the colors, but we can update each dynamically by subtracting off the old value of r_i and adding the new one, reducing the amortized cost per iteration from $O(n \log n)$ to $O(n)$. We still end up checking most of the edges, so the cost will now be $O\Big(n^{\lg k}(m+n)\Big)$. I don't see an obvious way to do better.

2. For this part we let $X_1, \ldots, X_n$ be the colors assigned to the vertices (ordering them arbitrarily) and let $f(X_1, \ldots, X_n)$ be the number of monochromatic edges given a particular coloring. Having fixed $X_1, \ldots, X_i$, we want to choose X_{i+1} to minimized $\mathrm{E}[f \mid X_1, \ldots, X_{i+1}]$.

   Observe that $f(X_1, \ldots, X_n) = \sum_{uv \in E}[X_u = X_v]$, so $\mathrm{E}[f | X_1, \ldots, X_i] = \sum_{uv \in E} \Pr[X_u = X_v]$. For each uv, if $i+1 \notin \{u, v\}$, then $[X_u = X_v]$ is independent of X_{i+1}, giving $\Pr[X_u = X_v \mid X_1, \ldots, X_{i+1}] = \Pr[X_u = X_v \mid X_1, \ldots, X_{i+1}]$. So we only care about the cases where $i+1 \in \{u, v\}$. Assume without loss of generality that $i+1 = u$. If $v \le i$, then $[X_u = X_v]$ is either 1 or 0 depending on whether we set $X_u = X_v$ or not. If $v > i+1$, then X_v is independent of X_{i+1}, so $\Pr[X_u = X_v \mid X_1, \ldots, X_{i+1}] = \Pr[X_u = X_v \mid X_1, \ldots, X_i]$.

   This means that to minimize $\mathrm{E}[f \mid X_1, \ldots, X_{i+1}]$, the only edges we need to consider are those edges X_{uv} where one endpoint is $i+1$ and the other endpoint is in $\{1, \ldots, i\}$. Minimizing the conditional expectation of f turns into the obvious greedy algorithm of picking a color for v_{i+1} that is least represented among its neighbors, which we can do in $O(1 + d(v_{i+1}))$ time. Adding up over all vertices v gives $O(\sum_{v \in V}(1 + d(v))) = O(n + m)$ time.

C.6.2 No long paths

Given a directed graph G with n vertices and maximum out-degree d, we would like to find an induced subgraph G' with at least $(1 - \delta)n/(d+1)$ vertices, such that G' has no simple paths[2] of length $\lceil 2d \ln n \rceil$ or greater.

Find an algorithm that does this in polynomial expected time for any constant $d > 0$ and $\delta > 0$, and sufficiently large n.

[2]For the purposes of this problem, a **simple path** of length ℓ is a directed path with ℓ edges and no repeated vertices.

Solution

We'll put each vertex v to G' with independent probability $p = 1/(d+1)$. We now have two possible bad outcomes:

1. We may get a path of length $\lceil 2d \ln n \rceil$ in G'.
2. We may get fewer than $(1-\delta)n/(d+1)$ vertices in G'.

We'll start by showing that the sum of the probabilities of these bad outcomes is not too big.

Let $\ell = \lceil 2d \ln n \rceil$. Observe that we can describe a path of length ℓ in G by specifying its starting vertex (n choices) and a sequence of ℓ edges each leaving the last vertex added so far (d choices each). This gives at most nd^ℓ paths of length ℓ in G. Each such path appears in G' with probability exactly $p^{\ell+1}$. Using the usual probabilistic-method argument, we get

$$\begin{aligned}
\Pr\left[G' \text{ includes at least one path of length } \ell\right] &\le \mathrm{E}\left[\text{number of paths of length } \ell \text{ in } G'\right] \\
&\le nd^\ell p^{\ell+1} \\
&= np(pd)^\ell. \\
&= \frac{1}{d+1} n \left(\frac{d}{d+1}\right)^\ell \\
&= \frac{1}{d+1} n \left(1 - \frac{1}{d+1}\right)^\ell \\
&= \frac{1}{d+1} n \left(1 - \frac{1}{d+1}\right)^{\lceil 2d \ln n \rceil} \\
&\le \frac{1}{d+1} n \left(1 - \frac{1}{d+1}\right)^{2d \ln n} \\
&\le \frac{1}{d+1} n e^{-\ln n} \\
&= \frac{1}{d+1}. \\
&\le \frac{1}{2}.
\end{aligned}$$

For the second-to-last inequality, we use the inequality $\left(1 - \frac{1}{d+1}\right)^{2d} \le e^{-1}$ for $d \ge 1$. This is easiest to demonstrate numerically, but if we had to prove it we could argue that it holds when $d = 1$ (since $1/4 < 1/e$) and that $(1 - \frac{1}{d+1})^{2d}$ is a decreasing function of d.

For getting too few vertices, use Chernoff bounds. Let X be the number of vertices in G'. Let $\mu = \mathrm{E}[X] = \frac{n}{d+1}$.

$$\begin{aligned}\Pr[X < (1-\delta)n/(d+1)] &= \Pr[X < (1-\delta)\mu] \\ &\le e^{-\mu\delta^2/2} \\ &= e^{-n\delta^2/(2(d+1))} \\ &< 1/4,\end{aligned}$$

for any fixed $d > 0$, $\delta > 0$, and sufficiently large n.

Adding these error probabilities together gives a total probability of failure of at most 3/4. We can select the vertices in G' in time linear in n. So if we can detect failure in polynomial time and try again, we only have to run at most 4 poly-time attempts on average until we win, giving polynomial expected cost.

So now let us detect failure. There are at most nd^ℓ paths of length ℓ in G. We can enumerate all of them and check if each is in G' in time

$$\begin{aligned}O\Big(nd^\ell\ell\Big) &= O\Big(n\ell d^{2d\ln n+1}\Big) \\ &= O\Big(n\ell de^{2d\ln n\ln d}\Big) \\ &= O\Big(n^{2d\ln d+1}d^2\log n\Big),\end{aligned}$$

which is polynomial in n for fixed d.

C.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are three problems on this exam, each worth 20 points, for a total of 60 points. You have approximately three hours to complete this exam.

C.7.1 A uniform ring

Suppose we have a ring of n nodes, each of which holds a bit. At each step, we pick a node uniformly at random and have it copy the bit of its left neighbor. For example, if we start in state 010, then picking the rightmost node will send us to state 011, and if we then pick the leftmost node we will reach state 111. From this last state no further changes are possible.

Suppose n is even and we start with a state where positions 1 through $n/2$ are all 0 and positions $n/2+1$ through n are all 1. What is the expected number of steps, as an asymptotic function of n, to reach a state where all bits are the same?

Solution

First observe that any transition preserves the property that our state looks like a string of the form 0^k1^{n-k}, where both the zeroes and ones all appear in consecutive positions around the ring.

Let X_t be the number of zeroes after t steps. We have $X_0 = n/2$, and our possible transitions when $0 < X_t < n$ are:

1. With probability $1/n$, we choose the leftmost zero and replace it with a one. This makes $X_{t+1} = X_t - 1$.

2. With probability $1/n$, we choose the leftmost one and replace it with a zero. This makes $X_{t+1} = X_t + 1$.

3. With the remaining probability $1 - 2/n$, we choose some node whose bit is equal to that of its left neighbor. This makes $X_{t+1} = X_t$.

When $X_t = 0$ or $X_t = n$, we have converged: no further changes are possible.

Looking at the transitions for X_t, we see that it is doing a very lazy random walk with absorbing barriers at 0 and n. If we remember that a non-lazy random walk converges in $O(n^2)$ steps on average, we can argue that the lazy version converges in $O(n^3)$ steps on average, because the expected waiting time between changes in X_t is $n/2 = O(n)$, and we can multiply this by the number of changes because of Wald's equation.

If we want to be very formal about this, we can also calculate the exact expected convergence time using the Optional Stopping Theorem. Let's guess that $Y_t = X_t^2 - ct$ is a martingale for some coefficient c. To find c, observe that we need

$$\begin{aligned}
\mathrm{E}\left[\left(X_{t+1}^2 - c(t+1)\right) - \left(X_t^2 - ct\right) \;\middle|\; X_t\right] &= \mathrm{E}\left[X_{t+1}^2 - X_t^2 - c \;\middle|\; X_t\right] \\
&= \frac{1}{n}(2X_t + 1) + \frac{1}{n}(-2X_t + 1) - c \\
&= \frac{2}{n} - c \\
&= 0.
\end{aligned}$$

So $\{Y_t\}$ is a martingale when $c = \frac{2}{n}$.

Let τ be the stopping time at which X_τ first equals 0 or n. The usual argument shows that $\mathrm{E}[\tau]$ is finite, and $\{Y_t\}$ has bounded increments because Y_t never changes by more than $2X_t + 1 \leq 2n + 1$. So OST applies and $\mathrm{E}[Y_\tau] = \mathrm{E}[Y_0] = (n/2)^2$. But we can expand

$$\begin{aligned}\mathrm{E}[Y_\tau] &= \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot n^2 - \frac{2}{n} \cdot \mathrm{E}[\tau] \\ &= \frac{n^2}{2} - \frac{2}{n} \cdot \mathrm{E}[\tau].\end{aligned}$$

This gives $\frac{n^2}{4} = \frac{n^2}{2} - \frac{2}{n} \cdot \tau$. Solving for $\mathrm{E}[\tau]$ gives $\mathrm{E}[\tau] = \frac{n^3}{8} = O(n^3)$.

C.7.2 Forbidden runs

Given a sequence of coin-flips, a **run** is a maximal subsequence of either all heads or all tails. The **length** of a run is the number of flips in the run. For example, the sequence HHHTTHTHH has five runs: a run HHH of length 3, a run TT of length 2, a run H of length 1, a run T of length 1, and a run HH of length 2.

Suppose we are attempting to transmit a sequence of n fair, independent coin-flips across a communication channel that uses a run of length k to signal an emergency condition. To avoid false alarms, we must encode our sequence to include no runs of length k. We do this by encoding any run of length $\ell \geq k$ as a run of length $\ell + 1$ of the same value. For example, when $k = 2$, we would encode the previous example HHHTTHTHH as HHHHTTTHTHHH.

As a function of n and k, what is the expected length of the encoded sequence?

Solution

Let i range from 1 to n, and let X_i be the i-th coin-flip in the sequence. Let R_i be the indicator variable for the event that there is a run of length k or more starting at position i. Then the expected number of runs of length k or more is given by $\sum_{i=1}^{n} \mathrm{E}[R_i]$ by linearity of expectation, and because each such run adds one to the length of the encoded sequence, the expected encoded length is $n + \sum_{i=1}^{n} \mathrm{E}[R_i]$.

To compute $\mathrm{E}[R_i]$, we consider three cases:

1. If $i = 1$, then $R_i = 1$ when $X_1, \ldots, X_k$ are all H or all T. This gives $\mathrm{E}[R_i] = 2 \cdot 2^{-k} = 2^{-k+1}$.

2. If $1 < i \leq n - k + 1$, then $R_i = 1$ when $X_1, \ldots, X_k$ are all H or all T, and X_{i-1} is not this common value (because otherwise the run starts earlier). This gives $\mathrm{E}[R_i] = 2 \cdot 2^{-k-1} = 2^{-k}$.

3. If $i > n - k + 1$, then $R_i = 0$, because there aren't enough coins left to get a run of length k.

Summing over all the cases gives

$$\begin{aligned}\sum_{i=1}^{n} \mathrm{E}[R_i] &= 2^{-k+1} + \sum_{i=2}^{n-k+1} 2^{-k} \\ &= 2 \cdot 2^{-k} + (n-k) \cdot 2^{-k} \\ &= (n-k+2) \cdot 2^{-k}.\end{aligned}$$

For the expected encoded sequence length, we must add back n to get $n + (n - k + 2) \cdot 2^{-k}$.

C.7.3 A derandomized balancing scheme

Suppose we construct an $n \times n$ matrix A where the elements A_{ij} are n^2 independent fair ± 1 random variables. Then Hoeffding's inequality tells us that

$$\Pr\left[\left|\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\right| \geq t\right] \leq 2\exp\left(-\frac{t^2}{2n^2}\right).$$

Unfortunately this requires using n^2 random bits.

As an alternative, suppose we use $2n$ random bits to generate two sequences X and Y of n independent fair ± 1 random variables, and define $B_{ij} = X_i Y_j$.

Show that

$$\Pr\left[\left|\sum_{i=1}^{n}\sum_{j=1}^{n} B_{ij}\right| \geq t\right] \leq 4\exp\left(-\frac{t}{2n}\right).$$

Solution

Start by factoring

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} B_{ij} &= \sum_{i=1}^{n}\sum_{j=1}^{n} X_i Y_j \\ &= \left(\sum_{i=1}^{n} X_i\right)\left(\sum_{j=1}^{n} Y_j\right).\end{aligned}$$

Each factor is a sum of n independent ± 1 random variables with expectation 0, so Hoeffding's inequality applies, giving

$$\Pr\left[\left|\sum_{i=1}^{n} X_i\right| \geq s\right] \leq 2\exp\left(-\frac{s^2}{2n}\right)$$

and

$$\Pr\left[\left|\sum_{j=1}^{n} Y_j\right| \geq s\right] \leq 2\exp\left(-\frac{s^2}{2n}\right).$$

If neither of these events holds, then $\left|\sum_{i=1}^{n}\sum_{j=1}^{n} B_{ij}\right| = \left|\sum_{i=1}^{n} X_i\right| \cdot \left|\sum_{j=1}^{n} Y_j\right| < s^2$.

So by the union bound,

$$\begin{aligned}\Pr\left[\left|\sum_{i=1}^{n}\sum_{j=1}^{n} B_{ij}\right| \geq s^2\right] &\leq \Pr\left[\left|\sum_{i=1}^{n} X_i\right| \geq s\right] + \Pr\left[\left|\sum_{j=1}^{n} Y_j\right| \geq s\right] \\ &\leq 4\exp\left(-\frac{s^2}{2n}\right).\end{aligned}$$

Now substitute t for s^2.

Appendix D

Sample assignments from Fall 2016

D.1 Assignment 1: due Sunday, 2016-09-18, at 17:00

Bureaucratic part

Send me email! My address is james.aspnes@gmail.com.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Whether you are taking the course as CPSC 469 or CPSC 569.
4. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

D.1.1 Bubble sort

Algorithm D.1 implements one pass of the (generally pretty slow) **bubble sort** algorithm. This involves comparing each array location with its successor, and swapping the elements if they are out of order. The full bubble sort

```
1 procedure BubbleSortOnePass(A, n)
2     for i ← 1 to n − 1 do
3         if A[i] > A[i + 1] then
4             Swap A[i] and A[i + 1]
```

Algorithm D.1: One pass of bubble sort

algorithm repeats this loop until the array is sorted, but here we just do it once.

Suppose that A starts out as a uniform random permutation of distinct elements. As a function of n, what is the exact expected number of swaps performed by Algorithm D.1?

Solution

The answer is $n - H_n$, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th harmonic number. There are a couple of ways to prove this. Below, we let A_i represent the original contents of $A[i]$, before doing any swaps. In each case, we will use the fact that after i iterations of the loop, $A[i]$ contains the largest of $A_i, \ldots, A_i$; this is easily proved by induction on i.

- Let X_{ij} be the indicator variable for the event that A_i is eventually swapped with A_j. For this to occur, A_i must be bigger than A_j, and must be present in $A[j-1]$ after $j-1$ passes through the loop. This happens if and only if A_i is the largest value in $A_1, \ldots, A_j$. Because these values are drawn from a uniform random permutation, by symmetry A_i is largest with probability exactly $1/j$. So $\mathrm{E}[X_{ij}] = 1/j$.

  Now sum X_{ij} over all pairs $i < j$. It is easiest to do this by summing

over j first:

$$\begin{aligned}
\mathrm{E}\left[\sum_{i<j} X_{ij}\right] &= \sum_{i<j} \mathrm{E}\left[X_{ij}\right] \\
&= \sum_{j=2}^{n} \sum_{i=1}^{j-1} \mathrm{E}\left[X_{ij}\right] \\
&= \sum_{j=2}^{n} \sum_{i=1}^{j-1} \frac{1}{j} \\
&= \sum_{j=2}^{n} \frac{j-1}{j} \\
&= \sum_{j=1}^{n} \frac{j-1}{j} \\
&= \sum_{j=1}^{n} (1 - \frac{1}{j}) \\
&= n - \sum_{j=1}^{n} \frac{1}{j} \\
&= n - H_n.
\end{aligned}$$

- Alternatively, let's count how many values are *not* swapped from $A[i]$ to $A[i-1]$. We can then subtract from n to get the number that are.

  Let Y_i be the indicator variable for the event that A_i is not swapped into $A[i-1]$. This occurs if, when testing $A[i-1]$ against $A[i]$, $A[i]$ is larger. Since we know that at this point $A[i-1]$ is the largest value among $A_1, \ldots, A_{i-1}$, $Y_i = 1$ if and only if A_i is greater than all of $A_1, \ldots, A_{i-1}$, or equivalently if A_i is the largest value in $A_1, \ldots, A_i$. Again by symmetry we have $\mathrm{E}\left[Y_i\right] = \frac{1}{i}$, and summing over all i gives an expected H_n values that are not swapped down. So there are $n - H_n$ values on average that are swapped down, which also gives the expected number of swaps.

D.1.2 Finding seats

A huge lecture class with n students meets in a room with $m \geq n$ seats. Because the room is dark, loud, and confusing, the students adopt the following seat-finding algorithm:

1. The students enter the room one at a time. No student enters until the previous student has found a seat.

2. To find a seat, a student chooses one of the m seats uniformly at random. If the seat is occupied, the student again picks a seat uniformly at random, continuing until they either find a seat or have made k attempts that led them to an occupied seat.

3. If a student fails to find a seat in k attempts, they contact an Undergraduate Seating Assistant, who leads them to an unoccupied seat.

Give an asymptotic (big-Θ) expression for the number of students who are helped by Undergraduate Seating Assistants, as a function of n, m, and k, where k is a constant independent of n and m. As with any asymptotic formula, this expression should work for large n and m; and should be as simple as possible.[1]

Solution

Define the i-th student to be the student who takes their seat after i students have already sat (note that we are counting from zero here, which makes things a little easier). Let X_i be the indicator variable for the even that the i-th student does not find a seat on their own and needs help from a USA. Each attempt to find a seat fails with probability i/m. Since each attempt is independent of the others, the probability that all k attempts fail is $(i/m)^k$.

The number of students who need help is $\sum_{i=0}^{n-1} X_i$, so the expected number is

$$\begin{aligned} \mathrm{E}\left[\sum_{i=0}^{n-1} X_i\right] &= \sum_{i=0}^{n-1} \mathrm{E}\left[X_i\right] \\ &= \sum_{i=0}^{n-1} (i/m)^k \\ &= m^{-k} \sum_{i=0}^{n-1} i^k. \end{aligned}$$

Unfortunately there is no simple expression for $\sum i^k$ that works for all k. However, we can approximate it from above and below using integrals.

[1] An earlier version of this problem allowed k to grow with n, which causes some trouble with the asymptotics.

From below, we have

$$\begin{aligned}\sum_{i=0}^{n-1} i^k &= \sum_{i=1}^{n-1} i^k \\ &\geq \int_0^{n-1} x^k dx \\ &= \left[\frac{1}{k+1}x^{k+1}\right]_0^{n-1} \\ &= \frac{1}{k+1}(n-1)^{k+1} \\ &= \Omega\Big(\frac{1}{k+1}n^{k+1}\Big).\end{aligned}$$

From above, an almost-identical computation gives

$$\begin{aligned}\sum_{i=0}^{n-1} i^k &\leq \int_0^{n} x^k dx \\ &= \left[\frac{1}{k+1}x^{k+1}\right]_0^{n} \\ &= frac1k+1n^(k+1) \\ &= O(\frac{1}{k+1}n^{k+1}).\end{aligned}$$

Combining the two cases gives that the sum is $\Theta\Big(\frac{1}{k+1}n^{k+1}\Big)$. We must also include m^{-k}, giving the full answer

$$\Theta\Big(\frac{1}{k+1}n^{k+1}m^{-k}\Big).$$

I like writing this as $\Theta\Big(n\frac{(n/m)^k}{k+1}\Big)$ to emphasize that the cost per student is $\Theta\Big(\frac{(n/m)^k}{k+1}\Big)$, but there are many different ways to write the same thing. Note that since we are dealing with asymptotics, we could also replace the $\frac{1}{k+1}$ with just $\frac{1}{k}$ and still be correct, but that seems less informative somehow.

D.2 Assignment 2: due Thursday, 2016-09-29, at 23:00

D.2.1 Technical analysis

It is well known that many human endeavors—including **sports reporting**, **financial analysis**, and **Dungeons and Dragons**—involve building narra-

tives on top of the output of a weighted random number generator [Mun11], a process that is now often automated [Hol16]. For this problem, we are going to consider a simplified model of automated financial analysis.

Let us model the changes in a stock price over n days using a sequence of independent ± 1 random variables $X_1, X_2, \ldots, X_n$, where each X_i is ± 1 with probability 1/2. Let $S_i = \sum_{j=1}^{i} X_j$, where $i \in \{0, \ldots, n\}$, be the price after i days.

Our automated financial reporter will declare a **dead cat bounce** if a stock falls in price for two days in a row, followed by rising in price, followed by falling in price again. Formally, a dead cat bounce occurs on day i if $i \geq 4$ and $S_{i-4} > S_{i-3} > S_{i-2} < S_{i-1} > S_i$. Let D be the number of dead cat bounces over the n days.

1. What is $\mathrm{E}[D]$?
2. What is $\mathrm{Var}[D]$?
3. Our investors will shut down our system if it doesn't declare at least one dead cat bounce during the first n days. What upper bound you can get on $\Pr[D = 0]$ using Chebyshev's inequality?
4. Show $\Pr[D = 0]$ is in fact exponentially small in n.

Note added 2016-09-28: It's OK if your solutions only work for sufficiently large n. This should save some time dealing with weird corner cases when n is small.

Solution

1. Let D_i be the indicator variable for the event that a dead cat bounce occurs on day i. Let $p = \mathrm{E}[D_i] = \Pr[D_i = 1]$. Then

$$p = \Pr[X_{i-3} = -1 \wedge X_{i-2} = -1 \wedge X_{i-1=+1} \wedge X_i = -1] = \frac{1}{16},$$

since the X_i are independent.

Then

$$\begin{aligned}\mathrm{E}[D] &= \mathrm{E}\left[\sum_{i=4}^{n} D_i\right] \\ &= \sum_{i=4}^{n} \mathrm{E}[D_i] \\ &= \sum_{i=4}^{n} \frac{1}{16} \\ &= \frac{n-3}{16},\end{aligned}$$

assuming n is at least 3.

2. For variance, we can't just sum up $\mathrm{Var}[D_i] = p(1-p) = \frac{15}{256}$, because the D_i are not independent. Instead, we have to look at covariance.

   Each D_i depends on getting a particular sequence of four values for X_{i-3}, X_{i-2}, X_{i-1}, and X_i. If we consider how D_i overlaps with D_j for $j > i$, we get these cases:

Variable	Pattern	Correlation
D_i	`--+-`	
D_{i+1}	`--+-`	inconsistent
D_{i+2}	`--+-`	inconsistent
D_{i+3}	`--+-`	overlap in one place

   For larger j, there is no overlap, so D_i and D_j are independent for $j \geq 4$, giving $\mathrm{Cov}[D_i, D_j] = 0$ for these cases.

   Because D_i can't occur with D_{i+1} or D_{i+2}, when $j \in \{i+1, i+2\}$ we have

$$\mathrm{Cov}[D_i, D_j] = \mathrm{E}[D_i D_j] - \mathrm{E}[D_i]\,\mathrm{E}[D_j] = 0 - (1/16)^2 = -1/256.$$

   When $j = i + 3$, the probability that both D_i and D_j occur is $1/2^7$, since we only need to get 7 coin-flips right. This gives

$$\mathrm{Cov}[D_i, D_{i+3}] = \mathrm{E}[D_i D_{i+3}] - \mathrm{E}[D_i]\,\mathrm{E}[D_{i+3}] = 1/128 - (1/16)^2 = 1/256.$$

Now apply (5.1.5):

$$\begin{aligned}
\operatorname{Var}[D] &= \operatorname{Var}\left[\sum_{i=4}^{n} D_i\right] \\
&= \sum_{i=4}^{n} \operatorname{Var}[D_i] + 2\sum_{i=4}^{n-1} \operatorname{Cov}[D_i, D_{i+1}] + 2\sum_{i=4}^{n-2} \operatorname{Cov}[D_i, D_{i+2}] + 2\sum_{i=4}^{n-3} \operatorname{Cov}[D_i, D_{i+3}] \\
&= (n-3)\frac{15}{256} + 2(n-4)\frac{-1}{256} + 2(n-5)\frac{-1}{256} + 2(n-6)\frac{1}{256} \\
&= \frac{13n-39}{256},
\end{aligned}$$

where to avoid special cases, we assume n is at least 6.

3. At this point we are just doing algebra. For $n \geq 6$, we have:

$$\begin{aligned}
\Pr[D=0] &\leq \Pr[|D - \operatorname{E}[D]| \geq \operatorname{E}[D]] \\
&= \Pr\left[|D - \operatorname{E}[D]| \geq \frac{n-3}{16}\right] \\
&\leq \frac{(13n-39)/256}{((n-3)/16)^2} \\
&= \frac{13n-39}{(n-3)^2} \\
&= \frac{13}{n-3}.
\end{aligned}$$

This is $\Theta(1/n)$, which is not an especially strong bound.

4. Here is an upper bound that avoids the mess of computing the exact probability, but is still exponentially small in n. Consider the events $[D_{4i} = 1]$ for $i \in \{1, 2, \ldots, \lfloor n/4 \rfloor\}$. These events are independent since they are functions of non-overlapping sequences of increments. So we can compute $\Pr[D=0] \leq \Pr[D_{4i} = 0, \forall i \in \{1, \ldots, \lfloor n/4 \rfloor\}] = (15/16)^{\lfloor n/4 \rfloor}$.

   This expression is a little awkward, so if we want to get an asymptotic estimate we can simplify it using $1 + x \leq e^x$ to get $(15/16)^{\lfloor n/4 \rfloor} \leq \exp(-(1/16)\lfloor n/4 \rfloor) = O(\exp(-n/64))$.

   With a better analysis, we should be able to improve the constant in the exponent; or, if we are real fanatics, calculate $\Pr[D=0]$ exactly. But this answer is good enough given what the problems asks for.

D.2.2 Faulty comparisons

Suppose we want to do a comparison-based sort of an array of n distinct elements, but an adversary has sabotaged our comparison subroutine. Specifically, the adversary chooses a sequence of times $t_1, t_2, \ldots, t_n$, such that for each i, the t_i-th comparison we do returns the wrong answer. Fortunately, this can only happen n times. In addition, the adversary is oblivious: thought it can examine the algorithm before choosing the input and the times t_i, it cannot predict any random choices made by the algorithm.

A simple deterministic work-around is to replace each comparison in our favorite $\Theta(n \log n)$-comparison sorting algorithm with $2n + 1$ comparisons, and take the majority value. But this gives us a $\Theta(n^2 \log n)$-comparison algorithm. We would like to do better using randomization.

Show that in this model, for every fixed $c > 0$ and sufficiently large n, it is possible to sort correctly with probability at least $1 - n^{-c}$ using $O(n \log^2 n)$ comparisons.

Solution

We'll adapt QuickSort. If the bad comparisons occurred randomly, we could just take the majority of $\Theta(\log n)$ faulty comparisons to simulate a non-faulty comparison with high probability. But the adversary is watching, so if we do these comparisons at predictable times, it could hit all $\Theta(\log n)$ of them and stay well within its fault budget. So we are going to have to be more sneaky.

Here's this idea: Suppose we have a list of n comparisons $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_n, y_n \rangle$ that we want to perform. We will use a subroutine that carries out these comparisons with high probability by doing $kn \ln n + n$ possibly-faulty comparisons, with k a constant to be chosen below, where each of the possibly-faulty comparisons looks at x_r and y_r where each r is chosen independently and uniformly at random. The subroutine collects the results of these comparisons for each pair $\langle x_i, y_i \rangle$ and takes the majority value.

At most n of these comparisons are faulty, and we get an error only if some pair $\langle x_i, y_i \rangle$ gets more faulty comparisons than non-faulty ones. Let B_i be the number of bad comparisons of the pair $\langle x_i, y_i \rangle$ and G_i the number of good comparisons. We want to bound the probability of the event $B_i > G_i$.

The probability that any particular comparison lands on a particular pair is exactly $1/n$; so $\mathrm{E}[B_i] = 1$ and $\mathrm{E}[G_i] = k \ln n$. Now apply Chernoff bounds. From (5.2.4), $\Pr[B_i \geq (k/2) \ln n] \leq 2^{-(k/2) \ln n} = n^{-(k \ln 2)/2)}$ provided $(k/2) \ln n$ is at least 6. In the other direction, (5.2.6) says that $\Pr[G_i \leq (k/2) \log n] \leq e^{-(1/2)^2 k \log n/2} = n^{-k/8}$. So we have a probability of

at most $n^{-k/2} + n^{-k/8}$ that we get the wrong result for this particular pair, and from the union bound we have a probability of at most $n^{1-k/2} + n^{1-k/8}$ that we get the wrong result for any pair. We can simplify this a bit by observe that n must be at least 2 (or we have nothing to sort), so we can put a bound of $n^{2-k/8}$ on the probability of any error among our n simulated comparisons, provided $k \geq 12/\ln 2$. We'll choose $k = \max(12/\ln 2, 8(c+3))$ to get a probability of error of at most n^{-c-1}.

So now let's implement QuickSort. The first round of QuickSort picks a pivot and performs $n-1$ comparisons. We perform these comparisons using our subroutine (note we can always throw in extra comparisons to bring the total up to n). Now we have two piles on which to run the algorithm recursively. Comparing all nodes in each pile to the pivot for that pile requires $n-3$ comparisons, which can again be done by our subroutine. At the next state, we have four piles, and we can again perform the $n-7$ comparisons we need using the subroutine. Continue until all piles have size 1 or less; this takes $\mathrm{O}(\log n)$ rounds with high probability. Since each round does $O(n \log n)$ comparisons and fails with probability at most n^{-c-1}, the entire process takes $O(n \log^2 n)$ comparisons and fails with probability at most $n^{-c-1}O(\log n) \leq n^{-c}$ when n is sufficiently large.

D.3 Assignment 3: due Thursday, 2016-10-13, at 23:00

D.3.1 Painting with sprites

In the early days of computing, memory was scarce, and primitive home game consoles like the **Atari 2600** did not possess enough RAM to store the image they displayed on your television. Instead, these machines used a graphics chip that would composite together fixed **sprites**, bitmaps stored in ROM that could be displayed at locations specified by the CPU.[2] One common programming trick of this era was to repurpose existing data in memory as sprites, for example by searching through a program's machine code instructions to find sequences of bytes that looked like explosions when displayed.

[2] In the actual Atari 2600 graphics hardware, the CPU did this not by writing down the locations in RAM somewhere (which would take memory), but by counting instruction cycles since the graphics chip started refreshing the screen and screaming "display this sprite NOW!" when the CRT beam got to the right position. For this problem we will assume a more modern approach, where we can just give a list of locations to the graphics hardware.

Figure D.1: Filling a screen with Space Invaders. The left-hand image places four copies of a 46-pixel sprite in random positions on a 40×40 screen. The right-hand image does the same thing with 24 copies.

For this problem, we will imagine that we have a graphics device that can only display one sprite. This is a bitmap consisting of m ones, at distinct relative positions $\langle y_1, x_1 \rangle, \langle y_2, x_2 \rangle, \ldots \langle y_m, x_m \rangle$. Displaying a sprite at position y, x on our $n \times n$ screen sets the pixels at positions $\langle (y + y_i) \bmod n, (x + x_i) \bmod n \rangle$ for all $i \in \{1, \ldots, m\}$. The screen is initially blank (all pixels 0) and setting a pixel at some position $\langle y, x \rangle$ changes it to 1. Setting the same pixel more than once has no effect.

We would like to use these sprites to simulate white noise on the screen, by placing them at independent uniform random locations with the goal of setting roughly half of the pixels. Unfortunately, because the contents of the screen are not actually stored anywhere, we can't detect when this event occurs. Instead, we want to fix the number of sprites ℓ so that we get $1/2 \pm o(1)$ of the total number of pixels set to 1 with high probability, by which we mean that $(1/2 \pm o(1))n^2$ total pixels are set with probability $1 - n^{-c}$ for any fixed $c > 0$ and sufficiently large n, assuming m is fixed.

An example of this process, using Taito Corporation's classic **Space Invader** bitmap, is shown in Figure D.1.

Compute the value of ℓ that we need as a function of n and m, and show that this choice of ℓ does in fact get $1/2 \pm o(1)$ of the pixels set with high probability.

Solution

We will apply the following strategy. First, we'll choose ℓ so that each individual pixel is set with probability close to $1/2$. Linearity of expectation then gives roughly $n^2/2$ total pixels set on average. To show that we get close to this number with high probability, we'll use the method of bounded differences.

The probability that pasting in a single copy of the sprite sets a particular pixel is exactly m/n^2. So the probability that the pixel is not set after ℓ sprites is $(1 - m/n^2)^\ell$. This will be exactly $1/2$ if $\ell = \log_{1-m/n^2} 1/2 = \frac{\ln(1/2)}{\ln(1-m/n^2)}$.

Since ℓ must be an integer, we can just round this quantity to the nearest integer to pick our actual ℓ. For example, when $n = 40$ and $m = 46$, $\frac{\ln(1/2)}{\ln(1-46/40^2)} = 23.7612...$ which rounds to 24.

For large n, we can use the approximation $\ln(1 + x) \approx x$ (which holds for small x) to approximate ℓ as $(n^2 \ln 2/m) + o(1)$. We will use this to get a concentration bound using (5.3.13).

We have ℓ independent random variables corresponding to the positions of the ℓ sprites. Moving a single sprite changes the number of set pixels by at most m. So the probability that the total number of set pixels deviates from its expectation by more than $an\sqrt{\ln n}$ is bounded by

$$\begin{aligned}
2\exp\left(-\frac{\left(an\sqrt{\ln n}\right)^2}{\ell m^2}\right) &= 2\exp\left(-\frac{a^2n^2\ln n}{(n^2\ln 2/m + o(1))m^2}\right) \\
&= 2\exp\left(-\frac{a^2\ln n}{m\ln 2 + o(m^2/n^2)}\right) \\
&\le \exp\Big(-(a^2/m)\ln n\Big) \\
&= n^{-a^2/m},
\end{aligned}$$

where the inequality holds when n is sufficiently large. Now choose $a = \sqrt{cm}$ to get a probability of deviating by more than $n\sqrt{cm\ln n}$ of at most n^{-c}. Since $n\sqrt{cm\ln n} = o(1)n^2$, this give us our desired bound.

D.3.2 Dynamic load balancing

A low-cost competitor to better-managed data center companies offers a scalable dynamic load balancing system that supplies a new machine whenever a new job shows up, so that there are always n machines available to run n jobs. Unfortunately, one of the ways the company cuts cost is by not

hiring programmers to replace the code from their previous randomized load balancing mechanism, so when the i-th job arrives, it is still assigned uniformly at random to one of the i available machines. This means that job 1 is always assigned to machine 1; job 2 is assigned with equal probability to machine 1 or 2; job 3 is assigned with equal probability to machine 1, 2, or 3; and so on. These choices are all independent. The company claims that the maximum load is still not too bad, and that the reduced programming costs that they pass on to their customers make up for their poor quality control and terrible service.

Justify the first part of this claim by showing that, with high probability, the most loaded machine after n jobs has arrived has load $O(\log n)$.

Solution

Let X_{ij} be the indicator variable for the event that machine i gets job j. Then $\mathrm{E}[X_{ij}] = 1/j$ for all $i \le j$, and $\mathrm{E}[X_{ij}] = 0$ when $i > j$.

Let $Y_i = \sum_{j=1}^{n} X_{ij}$ be the load on machine i.

Then $\mathrm{E}[Y_i] = \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}\right] = \sum_{j=i}^{n} 1/j \le \sum_{j=1}^{n} 1/j = H_n \le \ln n + 1$.

From (5.2.4), we have $\Pr[Y_i \ge R] \le 2^{-R}$ as long as $R > 2e\,\mathrm{E}[Y_i]$. So if we let $R = (c+1)\lg n$, we get a probability of at most n^{-c-1} that we get more than R jobs on machine i. Taking the union bound over all i gives a probability of at most n^{-c} that any machine gets a load greater than $(c+1)\lg n$. This works as long as $(c+1)\lg n \ge 2e(\ln n + 1)$, which holds for sufficiently large c. For smaller c, we can just choose a larger value c' that does work, and get that $\Pr[\max Y_i \ge (c'+1)\lg n] \le n^{-c'} \le n^{-c}$.

So for any fixed c, we get that with probability at least $1 - n^{-c}$ the maximum load is $O(\log n)$.

D.4 Assignment 4: due Thursday, 2016-11-03, at 23:00

D.4.1 Re-rolling a random treap

Suppose that we add to a random treap (see §6.3) an operation that re-rolls the priority of the node with a given key x. This replaces the old priority for x with a new priority generated uniformly at random, independently of the old priority any other priorities in the treap. We assume that the range of priorities is large enough that the probability that this produces a duplicate is negligible, and that the choice of which node to re-roll is done obliviously,

without regard to the current priorities of nodes or the resulting shape of the tree.

Changing the priority of x may break the heap property. To fix the heap property, we either rotate x up (if its priority now exceeds its parent's priority) or down (if its priority is now less than that of one or both of its children). In the latter case, we rotate up the child with the higher priority. We repeat this process until the heap property is restored.

Compute the best constant upper bound you can on the expected number of rotations resulting from executing a re-roll operation.

Solution

The best possible bound is at most 2 rotations on average in the worst case.

There is an easy but incorrect argument that 2 is an upper bound, which says that if we rotate up, we do at most the same number of rotations as when we insert a new element, and if we rotate down, we do at most the same number of rotations as when we delete an existing element. This gives the right answer, but for the wrong reasons: the cost of deleting x, conditioned on the event that re-rolling its priority gives a lower priority, is likely to be greater than 2, since the conditioning means that x is likely to be higher up in the tree than average; the same thing happens in the other direction when x moves up. Fortunately, it turns out that the fact that we don't necessarily rotate x all the way to or from the bottom compensates for this issue.

This can be formalized using the following argument, for which I am indebted to Adit Singha and Stanislaw Swidinski. Fix some element x, and suppose its old and new priorities are p and p'. If $p < p'$, we rotate up, and the sequence of rotations is exactly the same as we get if we remove all elements of the original treap with priority p or less and then insert a new element x with priority p'. But now if we condition on the number k of elements with priority greater than p, their priorities together with p' are all independent and identically distributed, since they are all obtained by taking their original distribution and conditioning on being greater than p. So all $(k+1)!$ orderings of these priorities are equally likely, and this means that we have the same expected cost as an insertion into a treap with k elements, which is at most 2. Averaging over all k shows that the expected cost of rotating up is at most 2, and, since rotating down is just the reverse of this process with a reversed distribution on priorities (since we get it by choosing p' as our old priority and p as the new one), the expected cost of rotating down is also at most 2. Finally, averaging the up and down cases gives that the expected number of rotations without conditioning on anything is at

most 2.

We now give an exact analysis of the expected number of rotations, which will show that 2 is in fact the best bound we can hope for.

The idea is to notice that whenever we do a rotation involving x, we change the number of ancestors of x by exactly one. This will always be a decrease in the number of ancestors if the priority of x went up, or an increase if the priority of x went down, so the total number of rotations will be equal to the change in the number of ancestors.

Letting A_i be the indicator for the event that i is an ancestor of x before the re-roll, and A'_i the indicator for the even that i is an ancestor of x after the re-roll, then the number of rotations is just $|\sum_i A_i - \sum_i A'_i|$, which is equal to $\sum_i |A_i - A'_i|$ since we know that all changes $A_i - A'_i$ have the same sign. So the expected number of rotations is just $\mathrm{E}\left[\sum_i |A_i - A'_i|\right] = \sum_i \mathrm{E}\left[|A_i - A'_i|\right]$, by linearity of expectation.

So we have to compute $\mathrm{E}\left[|A_i - A'_i|\right]$. Using the same argument as in §6.3.2, we have that $A_i = 1$ if and only if i has the highest initial priority of all elements in the range $[\min(i, x), \max(i, x)]$, and the same holds for A'_i if we consider updated priorities. So we want to know the probability that changing only the priority of x to a new random value changes whether i has the highest priority.

Let $k = \max(i, x) - \min(i, x) + 1$ be the number of elements in the range under consideration. To avoid writing a lot of mins and maxes, let's renumber these elements as 1 through k, with $i = 1$ and $x = k$ (this may involve flipping the sequence if $i > x$). Let $X_1, \ldots, X_k$ be the priorities of these elements, and let X'_k be the new priority of x. These $k + 1$ random variables are independent and identically distributed, so conditioning on the even that no two are equal, all $(k + 1)!$ orderings of their values are equally likely.

So now let us consider how many of these orderings result in $|A_i - A'_i| = 1$. For A_i to be 1, X_1 must exceed all of $X_2, \ldots, X_k$. For A'_i to be 0, X_1 must not exceed all of $X_2, \ldots, X_{k_1}, X'_k$. The intersection of these events is when $X'_k > X_1 > \max(X_2, \ldots, X_k)$. Since $X_2, \ldots, X_k$ can be ordered in any of $(k - 1)!$ ways, this gives

$$\Pr\left[A_i = 1 \wedge A'_i = 0\right] = \frac{(k-1)!}{(k+1)!} = \frac{1}{k(k+1)}.$$

In the other direction, for A_i to be 0 and A'_i to be 1, we must have $X_k >$

$X_1 > \max(X_1, \ldots, X_{k-1}, X_k)$. This again gives

$$\Pr\left[A_i = 0 \wedge A'_i = 1\right] = \frac{1}{k(k+1)}$$

and combining the two cases gives

$$\mathrm{E}\left[|A_i - A'_i|\right] = \frac{2}{k(k+1)}.$$

Now sum over all i:

$$\begin{aligned}
\mathrm{E}\left[\text{number of rotations}\right] &= \sum_{i=1}^{n} \mathrm{E}\left[|A'_i - A_i|\right] \\
&= \sum_{i=1}^{x-1} \frac{2}{(x-i+1)(x-i+2)} + \sum_{i=x+1}^{n} \frac{2}{(i-x+1)(i-x+2} \\
&= \sum_{k=2}^{x} \frac{2}{k(k+1)} + \sum_{k=2}^{n-x+1} \frac{2}{k(k+1)} \\
&= \sum_{k=2}^{x} \left(\frac{2}{k} - \frac{2}{k+1}\right) + \sum_{k=2}^{n-x+1} \left(\frac{2}{k} - \frac{2}{k+1}\right) \\
&= 2 - \frac{2}{x+1} - \frac{2}{n-x+2}.
\end{aligned}$$

In the limit as n goes to infinity, choosing $x = \lfloor n/2 \rfloor$ makes both fractional terms converge to 0, making the expected number of rotations arbitrarily close to 2. So 2 is the best possible bound that doesn't depend on n.

D.4.2 A defective hash table

A foolish programmer implements a hash table with m buckets, numbered $0, \ldots, m-1$, with the property that any value hashed to an even-numbered bucket is stored in a linked list as usual, but any value hashed to an odd-numbered bucket is lost.

1. Suppose we insert a set S of $n = |S|$ items into this hash table, using a hash function h chosen at random from a strongly 2-universal hash family H. Show that there is a constant c such that, for any $t > 0$, the probability that at least $n/2 + t$ items are lost is at most cn/t^2.

2. Suppose instead that we insert a set S of $n = |S|$ items into this hash table, using a hash function h chosen at random from a hash family

H' that is 2-universal, but that may not be strongly 2-universal. Show that if $n \leq m/2$, it is possible for an adversary that knows S to design a 2-universal hash family H' that ensures that all n items are lost with probability 1.

Solution

1. Let X_i be the indicator variable for the event that the i-th element of S is hashed to an odd-numbered bucket. Since H is strongly 2-universal, $\mathrm{E}[X_i] \leq 1/2$ (with equality when m is even), from which it follows that $\mathrm{Var}[X_i] = \mathrm{E}[X_i](1 - \mathrm{E}[X_i]) \leq 1/4$; and the X_i are pairwise independent. Letting $Y = \sum_{i=1}^{n} X_i$ be the total number of items lost, we get $\mathrm{E}[Y] \leq n/2$ and $\mathrm{Var}[Y] \leq n/4$. But then we can apply Chebyshev's inequality to show

$$\begin{aligned}\Pr[Y \geq n/2 + t] &\leq \Pr[Y \geq \mathrm{E}[Y] + t] \\ &\leq \frac{\mathrm{Var}[Y]}{t^2} \\ &\leq \frac{n/4}{t^2} \\ &= \frac{1}{4}(n/t^2).\end{aligned}$$

   So the desired bound holds with $c = 1/4$.

2. Write $[m]$ for the set $\{0, \ldots, m-1\}$. Let $S = \{x_1, x_2, \ldots, x_n\}$.

   Consider the set H' of all functions $h : U \to [m]$ with the property that $h(x_i) = 2i + 1$ for each i. Choosing a function h uniformly from this set corresponds to choosing a random function conditioned on this constraint; and this conditioning guarantees that every element of S is sent to an odd-numbered bucket and lost. This gives us half of what we want.

   The other half is that H' is 2-universal. Observe that for any $y \notin S$, the conditioning does not constrain $h(y)$, and so $h(y)$ is equally likely to be any element of $[m]$; in addition, $h(y)$ is independent of $h(z)$ for any $z \neq y$. So for any $y \neq z$, $\Pr[h(y) = h(z)] = 1/m$ if one or both of y and z is not an element of S, and $\Pr[h(y) = h(z)] = 0$ if both are elements of S. In either case, $\Pr[h(y) = h(z)] \leq 1/m$, and so H' is 2-universal.

D.5 Assignment 5: due Thursday, 2016-11-17, at 23:00

D.5.1 A spectre is haunting Halloween

An adult in possession of n pieces of leftover Halloween candy is distributing them to n children. The candies have known integer values $v_1, v_2, \ldots, v_n$, and the adult starts by permuting the candies according to a uniform random permutation π. They then give candy $\pi(1)$ to child 1, candy $\pi(2)$ to child 2, and so on. Or at least, that is their plan.

Unbeknownst to the adult, child n is a very young Karl Marx, and after observing the first k candies $\pi(1), \ldots, \pi(k)$, he can launch a communist revolution and seize the means of distribution. This causes the remaining $n - k$ candies $\pi(k + 1)$ through $\pi(n)$ to be distributed evenly among the remaining $n - k$ children, so that each receives a value equal to exactly $\frac{1}{n-k} \sum_{i=k+1}^{n} v_{\pi(i)}$. Karl may declare the revolution at any time before the last candy is distributed (even at time 0, when no candies have been distributed). However, his youthful understanding of the mechanisms of historical determinism are not detailed enough to predict the future, so his decision to declare a revolution after seeing k candies can only depend on those k candies and his knowledge of the values of all of the candies but not their order.

Help young Karl optimize his expected take by devising an algorithm that takes as input $v_1, \ldots, v_n$ and $v_{\pi(1)}, \ldots, v_{\pi(k)}$, where $0 \le k < n$, and outputs whether or not to launch a revolution at this time. Compute Karl's exact expected return as a function of $v_1, \ldots, v_n$ when running your algorithm, and show that no algorithm can do better.

Solution

To paraphrase an often-misquoted line of Trotsky's, young Karl Marx may not recognize the Optional Stopping Theorem, but the Optional Stopping Theorem does not permit him to escape its net. No strategy, no matter how clever, can produce an expected return better or worse than simply waiting for the last candy.

Let X_t be the expected return if Karl declares the revolution after seeing t cards. We will show that $\{X_t, \mathcal{F}_t\}$ is a martingale where each $\mathcal{F}_t$ is the σ-algebra generated by the random variables $v_{\pi(1)}$ through $v_{\pi(t)}$.

The value of X_t is exactly

$$X_t = \frac{1}{n-t}\sum_{i=t+1}^{n} v_{\pi(i)} = \frac{1}{n-t}\left(\sum_{i=t+1}^{n} v_n - \sum_{i=1}^{t} v_{\pi(i)}\right).$$

This is also $\mathrm{E}\left[v_{\pi(t+1)} \;\middle|\; \mathcal{F}_t\right]$, since the next undistributed candy is equally likely to be any of the remaining candies, and $\mathcal{F}_t$ determines the value of $v_{\pi(i)}$ for all $i \le t$. So

$$\begin{aligned}
\mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] &= \mathrm{E}\left[\frac{1}{n-t-1}\left(\sum_{i=1}^{n} v_i - \sum_{i=1}^{t+1} v_{\pi(i)}\right) \;\middle|\; \mathcal{F}_t\right] \\
&= \frac{1}{n-t-1}\left(\sum_{i=1}^{n} v_i - \sum_{i=1}^{t} v_{\pi(i)}\right) - \frac{1}{n-t-1}\mathrm{E}\left[v_{\pi(t+1)} \;\middle|\; \mathcal{F}_T\right] \\
&= \frac{n-t}{n-t-1}X_t - \frac{1}{n-t-1}X_t \\
&= X_t.
\end{aligned}$$

Now fix some strategy for Karl, and let τ be the time at which he launches the revolution. Then $\tau < n$ is a stopping time with respect to the $\mathcal{F}_t$, and the Optional Stopping Theorem (bounded time version) says that $\mathrm{E}[X_\tau] = \mathrm{E}[X_0]$. So any strategy is equivalent (in expectation) to launching the revolution immediately.

D.5.2 Colliding robots on a line

Suppose we have k robots on a line of n positions, numbered 1 through n. No two robots can pass each other or occupy the same position, so we can specify the positions of all the robots at time t as an increasing vector $X^t = \langle X_1^t, X_2^t, \ldots, X_k^t\rangle$. At each time t, we pick one of the robots i uniformly at random, and also choose a direction $d = \pm 1$ uniformly and independently at random. Robot i moves to position $X^{t+1} = X_i^t + d$ if (a) $0 \le X_i^t + d < n$, and (b) no other robot already occupies position $X_i^t + d$. If these conditions do not hold, robot i stays put, and $X_i^{t+1} = X_i^t$.

1. Given k and n, determine the stationary distribution of this process.

2. Show that the mixing time t_{mix} for this process is polynomial in n.

Solution

1. The stationary distribution is a uniform distribution on all $\binom{n}{k}$ placements of the robots. To prove this, observe that two vectors increasing vectors x and y are adjacent if and only if there is some i such that $x_j = y_j$ for all $j \neq i$ and $x_j + d = y_j$ for some $d \in \{-1, +1\}$. In this case, the transition probability p_{xy} is $\frac{1}{2n}$, since there is a $\frac{1}{n}$ chance that we choose i and a $\frac{1}{2}$ chance that we choose d. But this is the same as the probability that starting from y we choose i and $-d$. So we have $p_{xy} = p_{yx}$ for all adjacent x and y, which means that a uniform distribution π satisfies $\pi_x p_{xy} = \pi_y p_{yx}$ for all x and y.

   To show that this stationary distribution is unique, we must show that there is at least one path between any two states x and y. One way to do this is to show that there is a path from any state x to the state $\langle 1, \ldots, k \rangle$, where at each step we move the lowest-index robot i that is not already at position i. Since we can reverse this process to get to y, this gives a path between any x and y that occurs with nonzero probability.

2. This one could be done in a lot of ways. Below I'll give sketches of three possible approaches, ordered by increasing difficulty. The first reduces to card shuffling by adjacent swaps, the second uses an explicit coupling, and the third uses conductance.

   Of these approaches, I am personally only confident of the coupling argument, since it's the one I did before handing out the problem, and indeed this is the only one I have written up in enough detail below to be even remotely convincing. But the reduction to card shuffling is also pretty straightforward and was used in several student solutions, so I am convinced that it can be made to work as well. The conductance idea I am not sure works at all, but it seems like it could be made to work with enough effort.

   (a) Let's start with the easy method. Suppose that instead of colliding robots, we have a deck of n cards, of which k are specially marked. Now run a shuffling algorithm that swaps adjacent cards at each step. If we place a robot at the position of each marked card, the trajectories of the robots follow the pretty much the same distribution as in the colliding-robots process. This is trivially the case when we swap a marked card with an unmarked card (a robot moves), but it also works when we swap two marked cards

(no robot moves, since the positions of the set of marked cards stays the same; this corresponds to a robot being stuck).

Unfortunately we can't use exactly the same process we used in §10.4.3.3, because this (a) allows swapping the cards in the first and last positions of the deck, and (b) doesn't include any moves corresponding to a robot at position 1 or n trying to move off the end of the line.

The first objection is easily dealt with, and indeed the cited result of Wilson [Wil04] doesn't allow such swaps either. The second can be dealt with by adding extra no-op moves to the card-shuffling process that occur with probability $1/n$, scaling the probabilities of the other operations to keep the sum to 1. This doesn't affect the card-shuffling convergence argument much, but it is probably a good idea to check that everything still works.

Finally, even after fixing the card-shuffling argument, we still have to argue that convergence in the card-shuffling process implies convergence in the corresponding colliding-robot process. Here is where the definition of total variation distance helps. Let C^t be the permutation of the cards after t steps, and let $f : C^t \mapsto X^t$ map permutations of cards to positions of robots. Let π and π' be the stationary distributions of the card and robot processes, respectively. Then $d_{TV}(f(C^t), f(\pi)) = \max_A |\Pr[X^t \in A] - \pi'(A)| = \max_A |\Pr[C^t \in f^{-1}(A)] - \pi(f^{-1}(A))| \leq \max_B |\Pr[C^t \in B] - \pi(B)| = d_{TV}(C^t, \pi)$. So convergence of C implies convergence of $X = f(C)$.

(b) Alternatively, we can construct a coupling between two copies of the process X^t and Y^t, where as usual we start X^0 in our initial distribution (whatever it is) and Y^0 in the uniform stationary distribution. At each step we generate a pair (i, d) uniformly at random from $\{1, \ldots, n\} \times \{-1, +1\}$. Having generated this pair, we move robot i in direction d in both processes if possible. It is clear that once $X^t = Y^t$, it will continue to hold that $X^{t'} = Y^{t'}$ for all $t' > t$.

To show convergence we must show that X^t will in fact eventually hit Y^t. To do so, let us track $Z^t = \|X^t - Y^t\|_1 = \sum_{i=1}^k |X_i^t - Y_i^t|$. We will show that Z_t is a supermartingale with respect to the history $\mathcal{F}_t$ generated by the random variables $\langle X^s, Y^s \rangle$ for $s \leq t$. Specifically, we will show that at each step, the expected change in Z^t conditioning on X^t and Y^t is non-positive.

Fix X^t, and consider all $2n$ possible moves (i, d). Since (i, d)

doesn't change X_j^t for any $j \neq i$, the only change in Z^t will occur if one of X_i^t and Y_i^t can move and the other can't. There are several cases:

i. If $i = k$, $d = +1$, and exactly one of X_k^t and Y_k^t is n, then the copy of the robot not at n moves toward the copy at n, giving $Z_{t+1} - Z_t = -1$. The same thing occurs if $i = 1$, $d = -1$, and exactly one of X_1^t and Y_1^t is 1.
   It is interesting to note that these two cases will account for the entire nonzero part of $\mathrm{E}\left[Z_{t+1} - Z_t \mid \mathcal{F}_t\right]$, although we will not use this fact.

ii. If $i < k$, $d = +1$, $X_i^t + 1 = X_{i+1}^t$, but $Y_i^t + 1 < X_{i+1}^t$, and $X_i^t \leq Y_i^t$, then robot i can move right in Y^t but not in X^t. This gives $Z^{t+1} - Z^t = +1$.
   However, in this case the move $(i+1, -1)$ moves robot $i+1$ left in Y^t but not in X^t, and since $Y_{i+1}^t > Y_i^t + 1 > X_i^t = X_i^{t+1}$, this move gives $Z^{t+1} - Z^t = -1$. Since the moves $(i, +1)$ and $(i + 1, -1)$ occur with equal probability, these two changes in Z^t cancel out on average.
   Essentially the same analysis applies if we swap X^t with Y^t, and if we consider moves $(i, -1)$ where $i > 1$ in which one copy of the robot is blocked but not the other. If we enumerate all such cases, we find that for every move of these types that increase Z^t, there is a corresponding move that decreases Z^t; it follows that these changes sum to zero in expectation.

Because each X_i^t and Y_i^t lies in the range $[1, n]$, it holds trivially that $Z^t \leq k(n-1) < n^2$. Consider an unbiased ± 1 random walk W^t with a reflecting barrier at n^2 (meaning that at n^2 we always move down) and an absorbing barrier at 0. Then the expected time to reach 0 starting from $W^t = x$ is given by $x(2n^2 - x) \leq n^4$. Since this unbiased random walk dominates the biased random walk corresponding to changes in Z^t, this implies that Z^t can change at most n^4 times on average before reaching 0.

Now let us ask how often Z^t changes. The analysis above implies that there is a chance of at least $1/n$ that Z^t changes in any state where some robot i with $X_i^t \neq Y_i^t$ is blocked from moving freely in one or both directions in either the X^t or Y^t process. To show that this occurs after polynomially many steps, choose some i with $X_i^t \neq Y_i^t$ that is not blocked in either process. Then i is selected every $\frac{1}{n}$ steps on average, and it moves according to a ± 1 random

walk as long as it is not blocked. But a ± 1 random walk will reach position 1 or n in $O(n^2)$ steps on average (corresponding to $O(n^3)$ steps of the original Markov chain), and i will be blocked by the left or right end of the line if it is not blocked before then. It follows that we reach a state in which Z^t changes with probability at least $\frac{1}{n}$ after $O(n^3)$ expected steps, which means that Z^t changes every $O(n^4)$ on average. Since we have previously established that Z^t reaches 0 after $O(n^4)$ expected changes, this gives a total of $O(n^8)$ expected steps of the Markov process before $Z^t = 0$, giving $X^t = Y^t$. Markov's inequality and the Coupling Lemma then give $t_{\text{mix}} = O(n^8)$.

(c) I believe it may also be possible to show convergence using a conductance argument. The idea is to represent a state $\langle x_1, \ldots, x_k \rangle$ as the differences $\langle \Delta_1, \ldots, \Delta_n \rangle$, where $\Delta_1 = x_1$ and $\Delta_{i+1} = x_{i+1} - x_i$. This representation makes the set of possible states $\left\{ \Delta \;\middle|\; \Delta_i \geq 1, \sum_{i=1}^{k} \Delta_i \leq n \right\}$ look like a simplex, a pretty well-behaved geometric object. In principle it should be possible to show an isoperimetric inequality that the lowest-conductance sets in this simplex are the N points closest to any given corner, or possibly get a lower bound on conductance using canonical paths. But the Δ version of the process is messy (two coordinates change every time a robot moves), and there are some complication with this not being a lazy random walk, so I didn't pursue this myself.

D.6 Assignment 6: due Thursday, 2016-12-08, at 23:00

D.6.1 Another colliding robot

A warehouse consists of an $n \times n$ grid of locations, which we can think of as indexed by pairs (i, j) where $1 \leq i, j \leq n$. At night, the warehouse is patrolled by a robot executing a lazy random walk. Unfortunately for the robot, there are also m crates scattered about the warehouse, and if the robot attempts to walk onto one of the m grid locations occupied by a crate, it fails to do so, and instead emits a loud screeching noise. We would like to use the noises coming from inside the warehouse to estimate m.

Formally, if the robot is at position (i, j) at time t, then at time $t + 1$ it either (a) stays at (i, j) with probability 1/2; or (b) chooses a position (i', j') uniformly at random from $\{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$

and attempts to move to it. If the robot chooses not to move, it makes no noise. If it chooses to move, but (i', j') is off the grid or occupied by one of the m crates, then the robot stays at (i, j) and emits the noise. Otherwise it moves to (i', j') and remains silent. The robot's position at time 0 can be any unoccupied location on the grid.

To keep the robot from getting walled in somewhere, whatever adversary placed the crates was kind enough to ensure that if a crate was placed at position (i, j), then all of the eight positions

$$\begin{array}{ccc} (i-1, j+1) & (i, j+1) & (i+1, j+1) \\ (i-1, j) & & (i+1, j) \\ (i-1, j-1) & (i, j-1) & (i+1, j-1) \end{array}$$

reachable by a king's move from (i, j) are unoccupied grid locations. This means that they are not off the grid and not occupied by another crate, so that the robot can move to any of these eight positions.

Your job is to devise an algorithm for estimating m to within ϵ relative error with probability at least $1 - \delta$, based on the noises emitted by the robot. The input to your algorithm is the sequence of bits $x_1, x_2, x_3, \ldots$, where x_i is 1 if the robot makes a noise on its i-th step. Your algorithm should run in time polynomial in n, $1/\epsilon$, and $\log(1/\delta)$.

Solution

It turns out that ϵ is a bit of a red herring: we can in fact compute the *exact* number of crates with probability $1 - \delta$ in time polynomial in n and $\log(1/\delta)$.

The placement restrictions and laziness make this an irreducible aperiodic chain, so it has a unique stationary distribution π. It is easy to argue from reversibility that this is uniform, so each of the $N = n^2 - m$ unoccupied positions occurs with probability exactly $1/N$.

It will be useful to observe that we can assign three unique unoccupied positions to each crate, to the east, south, and southeast, and this implies $m \leq n^2/4$.

The idea now is to run the robot until the distribution on its position is close to π, and then see if it hits an obstacle on the next step. We can easily count the number of possible transitions that hit an obstacle, since there are $4m$ incoming edges to the crates, plus $4n$ incoming edges to the walls. Since each edge uv has probability $\pi_u p_{uv} = \frac{1}{8N}$ of being selected in the stationary distribution, the probability q that we hit an obstacle starting from π is exactly $\frac{n+m}{2N} = \frac{n+m}{n^2-m}$. This function is not trivial to invert, but we don't have to invert it: if we can compute its value (to some reasonable precision),

we can compare it to all $n^2/4 + 1$ possible values of m, and see which it matches. But we still have to deal with both getting enough samples and the difference between our actual distribution when we take a sample and the stationary distribution. For the latter, we need a convergence bound.

We will get this bound by constructing a family of canonical paths between each distinct pair of locations (i, j) and (i', j'). In the absence of crates, we can use the L-shaped paths that first change i to i' and then j to j', one step at a time in the obvious way. If we encounter a crate at some position along this path, we replace that position with a detour that uses up to three locations off the original path to get around the obstacle. The general rule is that if we are traveling left-to-right or right-to-left, we shift up one row to move past the obstacle, and if we are moving up-to-down or down-to-up, we shift left one column. An annoying special case is when the obstacle appears exactly at the corner (i', j); here we replace the blocked position with the unique diagonally adjacent position that still gives us a path.

Consider the number of paths $(i, j) \to (i', j')$ crossing a particular left-right edge $(x, y) \to (x + 1, y)$. There are two cases we have to consider:

1. We cross the edge as part of the left-to-right portion of the path (this includes left-to-right moves that are part of detours). In this case we have $|j - y| \leq 1$. This gives at most 3 choices for j, giving at most $3n^3$ possible paths. (The constants can be improved here.)

2. We cross the edge as part of a detour on the vertical portion of the path. Then $|i - x| \leq 1$, and so we again have at most $3n^3$ possible paths.

This gives at most $6n^3$ possible paths across each edge, giving a congestion

$$\begin{aligned}
\rho &\leq \frac{1}{\pi_{ij} p_{ij,i'j'}} (6n^3) \pi_{ij} \pi_{i'j'} \\
&= \frac{1}{N^{-1}(1/8)} (6n^3) N^{-2} \\
&= 24n^3 N^{-}1 \\
&\leq 24n^3 (\frac{3}{4} n^2)^{-1} \\
&= 32n,
\end{aligned}$$

since we can argue from the placement restrictions that $N \leq n^2/4$. This immediately gives a bound $\tau_2 \leq 8\rho^2 \leq O(n^2)$. using Lemma 10.6.3.

So let's run for $\lceil 51\tau_2 \ln n \rceil = O(n^2 \ln n)$ steps for each sample. Starting from any initial location, we will reach some distribution σ with $d_{TV}\sigma, \pi = O(n^{-50})$. Let X be the number of obstacles (walls or crates) adjacent to the current position, then we can apply Lemma 10.2.2 to get $|\mathrm{E}_\sigma(X) - \mathrm{E}_\pi(X)| \leq 4d_{TV}(\sigma, \pi) = O(n^{-50})$. The same bound (up to constants) also applies to the probability $\rho = X/8$ of hitting an obstacle, giving $\rho = 4(n+m)/(n^2 - m) \pm O(n^{-50})$. Note that ρ is $\Theta(n^{-1}$ for all values of m.

Now take $n^{10}\sqrt{\ln(1/\delta)}$ samples, with a gap of $\lceil 51\tau_2 \ln n \rceil$ steps between each. The expected number of positive samples is $\mu = (\rho + O(n^{-50}))n^{10}\sqrt{\ln(1/\delta)} = \Theta(n^9\sqrt{\ln(1/\delta)})$, and from Lemma 5.2.2, the probability that the number of positive samples exceeds $\mu(1+n^{-4})$ is at most $\exp(-\mu n^{-8}/3) = \exp(-\Theta(n \ln(1/\delta))) = \delta^{\Theta(n)} \leq \delta/2$ for sufficiently large n. A similar bound holds on the other side. So with probability at least $1-\delta$ we get an estimate $\hat{\rho}$ of $\rho = 4(n+m)/(n^2-m)$ that is accurate to within a relative error of n^{-4}. Since $\frac{d\rho}{dm} = O(n^{-2})$ throughout the interval spanned by m, and since ρ itself is $\Theta(n^{-1})$, any relative error that is $o(n^{-1})$ will give us an exact answer for m when we round to the nearest value of ρ. We are done.

We've chosen big exponents because all we care about is getting a polynomial bound. With a cleaner analysis we could probably get better exponents. It is also worth observing that the problem is ambiguous about whether we have random access to the data stream, and whether the time taken by the robot to move around counts toward our time bound. If we are processing the stream after the fact, and are allow to skip to specific places in the stream at no cost, we can get each sample in $O(1)$ time. This may further reduce the exponents, and demonstrates why it is important to nail down all the details of a model.

D.6.2 Return of the sprites

The Space Invaders from Problem D.3.1 are back, and now they are trying to hide in random $n \times n$ bitmaps, where each pixel is set with probability 1/2. Figure D.2 shows an example of this, with two sprites embedded in an otherwise random bitmap.

The sprites would like your assistance in making their hiding places convincing. Obviously it's impossible to disguise their presence completely, but to make the camouflage as realistic as possible, they would like you to provide them with an algorithm that will generate an $n \times n$ bitmap uniformly at random, conditioned on containing *at least two* copies of a given m-pixel sprite at distinct offsets. (It is OK for the sprites to overlap, but they cannot be directly on top of each other, and sprites that extend beyond the edges of

Figure D.2: Two hidden Space Invaders. On the left, the Space Invaders hide behind random pixels. On the right, their positions are revealed by turning the other pixels gray.

the bitmap wrap around as in Problem D.3.1.) Your algorithm should make the probability of each bitmap that contains at least two copies of the sprite be exactly equally likely, and should run in expected time polynomial in n and m.

Solution

Rejection sampling doesn't work here because if the sprites are large, the chances of getting two sprites out of a random bitmap are exponentially small. Generating a random bitmap and slapping two sprites on top of it also doesn't work, because it gives a non-uniform distribution: if the sprites overlap in k places, there are 2^{n^2-2m+k} choices for the remaining bits, which means that we would have to adjust the probabilities to account for the effect of the overlap. But even if we do this, we still have issues with bitmaps that contain more than two sprites: a bitmap with three sprites can be generated in three different ways, and it gets worse quickly as we generate more. It may be possible to work around these issues, but a simpler approach is to use the sampling mechanism from Karp-Luby [KL85] (see also §11.4).

Order all $\binom{n^2}{2}$ positions $u < v$ for the two planted sprites in lexicographic order. For each pair of positions $u < v$, let A_{uv} be the set of all bitmaps with sprites at u and v. Then we can easily calculate $|A_{uv}| = 2^{n^2-2m+k}$ where k is the number of positions where sprites at u and v overlap, and so we can sample a particular A_{uv} with probability $|A_{uv}|/\sum_{st}|A_{st}|$ and then choose an

element of A_{uv} uniformly at random by filling in the positions not covered by the sprites with independent random bits. To avoid overcounting, we discard any bitmap that contains a sprite at a position less than $\max(u, v)$; this corresponds to discarding any bitmap that appears in A_{st} for some $st < uv$. By the same argument as in Karp-Luby, we expect to discard at most $\binom{n^2}{2}$ bitmaps before we get one that works. This gives an expected $O(n^4)$ attempts, each of which requires about $O(n^2)$ work to generate a bitmap and $O(n^2 m)$ work to test it, for a nicely polynomial $O(n^6 m)$ expected time in total.

If we don't want to go through all this analysis, we could also reduce to Karp-Luby directly by encoding the existence of two sprites as a large DNF formula consisting of $\binom{n^2}{2}$ clauses, and then argue that the sampling mechanism inside Karp-Luby gives us what we want.

D.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are three problems on this exam, each worth 20 points, for a total of 60 points. You have approximately three hours to complete this exam.

D.7.1 Virus eradication (20 points)

Viruses have infected a network structured as an undirected graph $G = (V, E)$. At time 0, all $n = |V|$ nodes are infected. At each step, as long as there is at least one infected node, we can choose a node v to disinfect; this causes v to become uninfected, but there is an independent $\frac{1}{2d(v)}$ chance that each of v's neighbors becomes infected if it is not infected already, where $d(v)$ is the degree of v.

Let X_t be the number of nodes that are infected after t disinfection steps. Let τ be the first time at which $X_\tau = 0$. Show that, no matter what strategy is used to select nodes to disinfect, $\mathrm{E}[\tau] = O(n)$.

Clarification added during exam: When choosing a node to disinfect, the algorithm can only choose nodes that are currently infected.

Solution

Let $Y_t = X_t + \frac{1}{2}\min(t, \tau)$. We will show that $\{Y_t\}$ is a supermartingale. Suppose we disinfect node v at time t, and that v has $d' \le d(v)$ uninfected neighbors. Then $\mathrm{E}[X_{t+1} \mid X_t] = X_t - 1 + \frac{d'}{2d(v)} \le X_t - \frac{1}{2}$, because we always

disinfect v and each of its d' uninfected neighbors contributes $\frac{1}{2d(v)}$ new infections on average. We also have $\min(\tau, t)$ increasing by 1 in this case. This gives $\mathrm{E}[Y_{t+1} \mid Y_t] \le Y_t - \frac{1}{2} + \frac{1}{2} = Y_t$ when $t < \tau$; it also holds trivially that $Y_{t+1} = Y_t$ when $t \ge \tau$. So $\mathrm{E}[Y_{t+1} \mid Y_t] \le Y_t$ always.

Now apply the optional stopping theorem. We have bounded increments, because $|Y_{t+1} - Y_t| \le n + \frac{1}{2}$. We also have bounded expected time, because there is a nonzero probability ϵ that each sequence of n consecutive disinfections produces no new infections, giving $\mathrm{E}[\tau] \le 1/\epsilon$. So $n = Y_0 \ge \mathrm{E}[Y_\tau] = \frac{1}{2}\mathrm{E}[\tau]$, giving $\mathrm{E}[\tau] \le 2n = O(n)$.

D.7.2 Parallel bubblesort (20 points)

Suppose we are given a random bit-vector and choose to sort it using parallel bubblesort. The initial state consists of n independent fair random bits $X_1^0, X_2^0, \ldots, X_n^0$, and in each round, for each pair of positions X_i^t and X_{i+1}^t, we set new values $X_i^{t+1} = 0$ and $X_{i+1}^{t+1} = 1$ if $X_i^t = 1$ and $X_{i+1}^t = 0$. When it holds that $X_i^t = 1$ and $X_{i+1}^t = 0$, we say that there is a swap at position i in round t. Denote the total number of swaps in round t by Y_t.

1. What is $\mathrm{E}[Y_0]$?

2. What is $\mathrm{E}[Y_1]$?

3. Show that both Y_0 and Y_1 are concentrated around their expectations, by showing, for any fixed $c > 0$ and each $t \in \{0, 1\}$, that there exists a function $f_t(n) = o(n)$ such that $\Pr[|Y_t - \mathrm{E}[Y_t]| \ge f_t(n)] \le n^{-c}$.

Solution

1. This is a straightforward application of linearity of expectation. Let Z_i^t, for each $i \in \{1, \ldots, n-1\}$, be the indicator for the event that $X_i^t = 1$ and $X_{i+1}^t = 0$. For $t = 0$, X_i^0 and X_{i+1}^0 are independent, so this event occurs with probability $\frac{1}{4}$. So $\mathrm{E}[Y_0] = \mathrm{E}\left[\sum_{i=1}^{n-1} Z_i^0\right] = \sum_{i=1}^{n-1} \mathrm{E}[Z_i^0] = (n-1)\frac{1}{4} = \frac{n-1}{4}$.

2. Again we want to use linearity of expectation, but we have to do a little more work to calculate $\mathrm{E}[Z_i^1]$. For Z_i^1 to be 1, we need $X_i^1 = 1$ and $X_{i+1}^1 = 0$. If we ask where that 1 in X_i^1 came from, either it started in X_{i-1}^0 and moved to position i because $X_i^0 = 0$, or it started in X_i^0 and didn't move because $X_{i+1}^0 = 1$. Similarly the 0 in X_{i+1}^1 either started in X_{i+2}^0 and moved down to replace $X_{i+1}^0 = 1$ or started in X_{i+1}^1 and

didn't move because $X_i^0 = 0$. Enumerating all the possible assignments of X_{i-1}^0, X_i^0, X_{i+1}^0, and X_{i+2}^0 consistent with these conditions gives 0110, 1000, 1001, 1010, and 1110, making $\mathrm{E}\left[Z_i^1\right] = 5 \cdot 2^{-4} = \frac{5}{16}$ when $2 \le i \le n-2$. This accounts for $n-3$ of the positions.

There are two annoying special cases: when $i = 1$ and when $i = n-1$. We can handle these with the above analysis by pretending that $X_0^t = 0$ and $X_{n+1}^t = 1$ for all t; this leaves the initial patterns 0110 for $i = 1$ and 1001 for $i = n-1$, each of which occurs with probability $\frac{1}{8}$.

Summing over all cases then gives $\mathrm{E}[Y_1] = \frac{1}{8} + \sum_{i=2}^{n-3} \frac{5}{16} + \frac{1}{8} = \frac{5n-11}{16}$.

3. The last part is just McDiarmid's inequality. Because the algorithm is deterministic aside from the initial random choice of input, We can express each Y_t as a function $g_t(X_1^0, \ldots, X_n^0)$ of the independent random variables $X_1^0, \ldots, X_n^0$. Changing some X_i^0 can change Z_{i-1}^0 and Z_i^0, for a total change to Y_0 of at most 2; similarly, changing X_i^0 can change at most Z_{i-2}^1, Z_{i-1}^1, Z_i^1, and Z_{i+1}^i, for a total change of at most 4. So McDiarmid says that, for each $t \in \{0, 1\}$, $\Pr\left[|Y_t - \mathrm{E}[Y_t]| \le s\right] \le 2\exp -\Theta(s^2/n)$. For any fixed $c > 0$, there is some $s = \Theta(\sqrt{n \log n}) = o(n)$ such that the right-hand side is less than n^{-c}.

D.7.3 Rolling a die (20 points)

Suppose we have a device that generates a sequence of independent fair random coin-flips, but what we want is a six-sided die that generates the values $1, 2, 3, 4, 5, 6$ with equal probability.

1. Give an algorithm that does so using $O(1)$ coin-flips on average.

2. Show that any correct algorithm for this problem will use more than n coin-flips with probability at least $2^{-O(n)}$.

Solution

1. Use rejection sampling: generate 3 bits, and if the resulting binary number is not in the range $1, \ldots, 6$, try again. Each attempt consumes 3 bits and succeeds with probability 3/4, so we need to generate $\frac{4}{3} \cdot 3 = 4$ bits on average.

   It is possible to improve on this by reusing the last bit of a discarded triple of bits as the first bit in the next triple. This requires a more complicated argument to show uniformity, but requires only two bits

for each attempt after the first, for a total of $3 + \frac{1}{4} \cdot \frac{4}{3} \cdot 2 = \frac{11}{3}$ bits on average. This is still $O(1)$, so unless that $\frac{1}{3}$ bit improvement is really important, it's probably easiest just to do rejection sampling.

2. Suppose that we have generated n bits so far. Since 6 does not evenly divide 2^n for any n, we cannot assign an output from $1, \ldots, 6$ to all possible 2^n sequences of bits without giving two outputs different probabilities. So we must keep going in at least one case, giving a probability of at least $2^{-n} = 2^{-O(n)}$ that we continue.

Appendix E

Sample assignments from Spring 2014

E.1 Assignment 1: due Wednesday, 2014-09-10, at 17:00

E.1.1 Bureaucratic part

Send me email! My address is james.aspnes@gmail.com.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

E.1.2 Two terrible data structures

Consider the following data structures for maintaining a sorted list:

1. A sorted array A. Inserting a new element at position $A[i]$ into an array that current contains k elements requires moving the previous values in $A[i] \dots A[k]$ to $A[i+1] \dots A[k+1]$. Each element moved costs one unit.

For example, if the array currently contains `1 3 5`, and we insert `2`, the resulting array will contain `1 2 3 5` and the cost of the operation will be 2, because we had to move `3` and `5`.

2. A sorted doubly-linked list. Inserting a new element at a new position requires moving the head pointer from its previous position to a neighbor of the new node, and then to the new node; each pointer move costs one unit.

   For example, if the linked list currently contains `1 3 5`, with the head pointer pointing to `5`, and we insert `2`, the resulting linked list will contain `1 2 3 5`, with the head pointing to `2`, and the cost will be 2, because we had to move the pointer from `5` to `3` and then from `3` to `2`. Note that we do not charge for updating the pointers between the new element and its neighbors. We will also assume that inserting the first element is free.

Suppose that we insert the elements 1 through n, in random order, into both data structures. Give an exact closed-form expression for the expected cost of each sequence of insertions.

Solution

1. Suppose we have already inserted k elements. Then the next element is equally likely to land in any of the positions $A[1]$ through $A[k+1]$. The number of displaced elements is then uniformly distributed in 0 through k, giving an expected cost for this insertion of $\frac{k}{2}$.

   Summing over all insertions gives

$$\begin{aligned}\sum_{k=0}^{n-1} \frac{k}{2} &= \frac{1}{2}\sum_{k=0}^{n-1} \\ &= \frac{n(n-1)}{4}.\end{aligned}$$

   An alternative proof, which also uses linearity of expectation, is to define X_{ij} as the indicator variable for the event that element j moves when element $i < j$ is inserted. This is 1 if an only if j is inserted before i, which by symmetry occurs with probability exactly $1/2$. So

the expected total number of moves is

$$\begin{aligned}\sum_{1\le i<j\le n} \mathrm{E}\left[X_{ij}\right] &= \sum_{1\le i<j\le n} \frac{1}{2} \\ &= \frac{1}{2}\binom{n}{2} \\ &= \frac{n(n-1)}{4}.\end{aligned}$$

It's always nice when we get the same answer in situations like this.

2. Now we need to count how far the pointer moves between any two consecutive elements. Suppose that we have already inserted $k-1>0$ elements, and let X_k be the cost of inserting the k-th element. Let i and j be the indices in the sorted list of the new and old pointer positions after the k-th insertion. By symmetry, all pairs of distinct

positions $i \neq j$ are equally likely. So we have

$$\begin{aligned}
\mathrm{E}[X_k] &= \frac{1}{k(k-1)} \sum_{i \neq j} |i-j| \\
&= \frac{1}{k(k-1)} \left(\sum_{1 \leq i < j \leq k} (j-i) + \sum_{1 \leq j < i \leq k} (i-j) \right) \\
&= \frac{2}{k(k-1)} \sum_{1 \leq i < j \leq k} (j-i) \\
&= \frac{2}{k(k-1)} \sum_{j=1}^{k} \sum_{\ell=1}^{j-1} \ell \\
&= \frac{2}{k(k-1)} \sum_{j=1}^{k} \frac{j(j-1)}{2} \\
&= \frac{1}{k(k-1)} \sum_{j=1}^{k} j(j-1). \\
&= \frac{1}{k(k-1)} \sum_{j=1}^{k} \left(j^2 - j \right) \\
&= \frac{1}{k(k-1)} \cdot \left(\frac{(2k+1)k(k+1)}{6} - \frac{k}{k+1} 2 \right) \\
&= \frac{1}{k(k-1)} \cdot \frac{(2k-2)k(k+1)}{6} \\
&= \frac{k+1}{3}.
\end{aligned}$$

This is such a simple result that we might reasonably expect that there is a faster way to get it, and we'd be right. A standard trick is to observe that we can simulate choosing k points uniformly at random from a line of n points by instead choosing $k+1$ points uniformly at random from a cycle of $n+1$ points, and deleting the first point chosen to turn the cycle back into a line. In the cycle, symmetry implies that the expected distance between each point and its successor is the same as for any other point; there are $k+1$ such distances, and they add up to $n+1$, so each expected distance is exactly $\frac{n+1}{k+1}$.

In our particular case, n (in the formula) is k and k (in the formula) is 2, so we get $\frac{k+1}{3}$. Note we are sweeping the whole absolute value thing under the carpet here, so maybe the more explicit derivation is safer.

However we arrive at $\mathrm{E}[X_k] = \frac{k+1}{3}$ (for $k > 1$), we can sum these expectations to get our total expected cost:

$$\begin{aligned}
\mathrm{E}\left[\sum_{k=2}^{n} X_k\right] &= \sum_{k=2}^{n} \mathrm{E}[X_k] \\
&= \sum_{k=2}^{n} \frac{k+1}{3} \\
&= \frac{1}{3}\sum_{\ell=3}^{n+1} \ell \\
&= \frac{1}{3}\left(\frac{(n+1)(n+2)}{2} - 3\right) \\
&= \frac{(n+1)(n+2)}{6} - 1.
\end{aligned}$$

It's probably worth checking a few small cases to see that this answer actually makes sense.

For large n, this shows that the doubly-linked list wins, but not by much: we get roughly $n^2/6$ instead of $n^2/4$. This is a small enough difference that in practice it is probably dominated by other constant-factor differences that we have neglected.

E.1.3 Parallel graph coloring

Consider the following algorithm for assigning one of k colors to each node in a graph with m edges:

1. Assign each vertex u a color c_u, chosen uniformly at random from all k possible colors.

2. For each vertex u, if u has a neighbor v with $c_u = c_v$, assign u a new color c'_u, again chosen uniformly at random from all k possible colors. Otherwise, let $c'_u = c_u$.

Note that any new colors c' do not affect the test $c_u = c_v$. A node changes its color only if it has the same original color as the original color of one or more of its neighbors.

Suppose that we run this algorithm on an r-regular[1] triangle-free[2] graph. As a function of k, r, and m, give an exact closed-form expression for the

[1] Every vertex has exactly r neighbors.

[2] There are no vertices u, v, and w such that all are neighbors of each other.

expected number of monochromatic[3] edges after running both steps of the algorithm.

Solution

We can use linearity of expectation to compute the probability that any particular edge is monochromatic, and then multiply by m to get the total.

Fix some edge uv. If either of u or v is recolored in step 2, then the probability that $c'_u = c'_v$ is exactly $1/k$. If neither is recolored, the probability that $c'_u = c'_v$ is zero (otherwise $c_u = c_v$, forcing both to be recolored). So we can calculate the probability that $c'_u = c'_v$ by conditioning on the event A that neither vertex is recolored.

This event occurs if both u and v have no neighbors with the same color. The probability that $c_u = c_v$ is $1/k$. The probability that any particular neighbor w of u has $c_w = c_u$ is also $1/k$; similarly for any neighbor w of v. These events are all independent on the assumption that the graph is triangle-free (which implies that no neighbor of u is also a neighbor of v). So the probability that none of these $2r-1$ events occur is $(1-1/k)^{2r-1}$.

We then have

$$\begin{aligned}\Pr[c_u = c_v] &= \Pr\left[c_u = c_v \mid \overline{A}\right]\Pr\left[\overline{A}\right] + \Pr[c_u = c_v \mid A]\Pr[A] \\ &= \frac{1}{k}\cdot\left(1-\left(1-\frac{1}{k}\right)^{2r-1}\right).\end{aligned}$$

Multiply by m to get

$$\frac{m}{k}\cdot\left(1-\left(1-\frac{1}{k}\right)^{2r-1}\right).$$

For large k, this is approximately $\frac{m}{k}\cdot\left(1-e^{-(2r-1)/k}\right)$, which is a little bit better than than the $\frac{m}{k}$ expected monochrome edges from just running step 1.

Repeated application of step 2 may give better results, particular if k is large relative to r. We will see this technique applied to a more general class of problems in §13.3.5.

[3] Both endpoints have the same color.

E.2 Assignment 2: due Wednesday, 2014-09-24, at 17:00

E.2.1 Load balancing

Suppose we distribute n processes independently and uniformly at random among m machines, and pay a communication cost of 1 for each pair of processes assigned to different machines. Let C be the total communication cost, that is, the number of pairs of processes assigned to different machines. What is the expectation and variance of C?

Solution

Let C_{ij} be the communication cost between machines i and j. This is just an indicator variable for the event that i and j are assigned to different machines, which occurs with probability $1 - \frac{1}{m}$. We have $C = \sum_{1 \le i < j \le n} C_{ij}$.

1. Expectation is a straightforward application of linearity of expectation. There are $\binom{n}{2}$ pairs of processes, and $\mathrm{E}[C_{ij}] = 1 - \frac{1}{m}$ for each pair, so
$$\mathrm{E}[C] = \binom{n}{2}\left(1 - \frac{1}{m}\right).$$

2. Variance is a little trickier because the C_{ij} are not independent. But they are pairwise independent: even if we fix the location of i and j, the expectation of C_{jk} is still $1 - \frac{1}{m}$, so $\mathrm{Cov}[C_{ij}, C_{jk}] = \mathrm{E}[C_{ij}C_{jk}] - \mathrm{E}[C_{ij}] \cdot \mathrm{E}[C_{jk}] = 0$. So we can compute
$$\mathrm{Var}[C] = \sum_{1 \le i < j < n} \mathrm{Var}[C_{ij}] = \binom{n}{2}\frac{1}{m}\left(1 - \frac{1}{m}\right).$$

E.2.2 A missing hash function

A clever programmer inserts X_i elements in each of m buckets in a hash table, where each bucket i is implemented as a balanced binary search tree with search cost at most $\lceil \lg(X_i + 1) \rceil$. We are interested in finding a particular target element x, which is equally likely to be in any of the buckets, but we don't know what the hash function is.

Suppose that we know that the X_i are independent and identically distributed with $\mathrm{E}[X_i] = k$, and that the location of x is independent of the values of the X_i. What is best upper bound we can put on the expected cost of finding x?

Solution

The expected cost of searching bucket i is $\mathrm{E}\left[\lceil \lg(X_i + 1) \rceil\right]$. This is the expectation of a function of X_i, so we would like to bound it using Jensen's inequality (§4.3).

Unfortunately the function $f(n) = \lceil \lg(n+1) \rceil$ is not concave (because of the ceiling), but $1 + \lg(n+1) > \lceil \lg(n+1) \rceil$ is. So the expected cost of searching bucket i is bounded by $1 + \lg(\mathrm{E}\left[X_i\right] + 1) = 1 + \lg(k+1)$.

Assuming we search the buckets in some fixed order until we find x, we will search Y buckets where $\mathrm{E}\left[Y\right] = \frac{n+1}{2}$. Because Y is determined by the position of x, which is independent of the X_i, Y is also independent of the X_i. So Wald's equation (3.4.3) applies, and the total cost is bounded by

$$\frac{n+1}{2}\left(1 + \lg(k+1)\right).$$

E.3 Assignment 3: due Wednesday, 2014-10-08, at 17:00

E.3.1 Tree contraction

Suppose that you have a tree data structure with n nodes, in which each node i has a pointer $\mathsf{parent}(u)$ to its parent (or itself, in the case of the root node root).

Consider the following randomized algorithm for shrinking paths in the tree: in the first phase, each node u first determines its parent $v = \mathsf{parent}(u)$ and its grandparent $w = \mathsf{parent}(\mathsf{parent}(u))$. In the second phase, it sets $\mathsf{parent}'(u)$ to v or w according to a fair coin-flip independent of the coin-flips of all the other nodes.

Let T be the tree generated by the parent pointers and T' the tree generated by the parent' pointers. An example of two such trees is given in Figure E.1.

Recall that the **depth** of a node is defined by $\mathrm{depth}(\mathsf{root}) = 0$ and $\mathrm{depth}(u) = 1 + \mathrm{depth}(\mathsf{parent}(u))$ when $u \neq \mathsf{root}$. The depth of a tree is equal to the maximum depth of any node.

Let D be depth of T, and D' be the depth of T'. Show that there is a constant a, such that for any fixed $c > 0$, with probability at least $1 - n^{-c}$ it holds that

$$a \cdot D - O\left(\sqrt{D \log n}\right) \leq D' \leq a \cdot D + O\left(\sqrt{D \log n}\right). \tag{E.3.1}$$

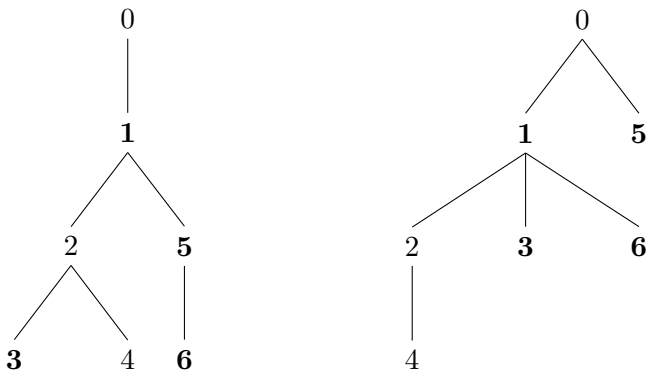


Figure E.1: Example of tree contraction for Problem E.3.1. Tree T is on the left, T' on the right. Nodes 1, 3, 5, and 6 (in boldface) switch to their grandparents. The other nodes retain their original parents.

Solution

For any node u, let $\text{depth}(u)$ be the depth of u in T and $\text{depth}'(u)$ be the depth of u in T'. Note that $\text{depth}'(u)$ is a random variable. We will start by computing $\mathrm{E}\left[\text{depth}'(u)\right]$ as a function of $\text{depth}(u)$, by solving an appropriate recurrence.

Let $S(k) = \text{depth}'(u)$ when $\text{depth}(u) = k$. The base cases are $S(0) = 0$ (the depth of the root never changes) and $S(1) = 1$ (same for the root's children). For larger k, we have

$$\mathrm{E}\left[\text{depth}'(u)\right] = \frac{1}{2}\mathrm{E}\left[1 + \text{depth}'(\mathsf{parent}(u))\right] + \frac{1}{2}\mathrm{E}\left[1 + \text{depth}'(\mathsf{parent}(\mathsf{parent}(u)))\right]$$

or

$$S(k) = 1 + \frac{1}{2}S(k-1) + \frac{1}{2}S(k-2).$$

There are various ways to solve this recurrence. The most direct may be to define a generating function $F(z) = \sum_{k=0}^{\infty} S(k)z^k$. Then the recurrence becomes

$$F = \frac{z}{1-z} + \frac{1}{2}zF + \frac{1}{2}z^2F.$$

Solving for F gives

$$\begin{aligned} F &= \frac{\frac{z}{1-z}}{1-\frac{1}{2}z-\frac{1}{2}z^2} \\ &= \frac{2z}{(1-z)(2-z-z^2)}. \\ &= \frac{2z}{(1-z)^2(2+z)} \\ &= 2z\left(\frac{1/3}{(1-z)^2}+\frac{1/9}{(1-z)}+\frac{1/18}{1+\frac{1}{2}z}\right) \\ &= \frac{2}{3}\cdot\frac{z}{(1-z)^2}+\frac{2}{9}\cdot\frac{z}{1-z}+\frac{1}{9}\cdot\frac{z}{1+\frac{1}{2}z}, \end{aligned}$$

from which we can read off the exact solution

$$S(k) = \frac{2}{3}\cdot k+\frac{2}{9}+\frac{1}{9}\left(-\frac{1}{2}\right)^{k-1}$$

when $k \geq 1$.[4]

We can easily show that $\text{depth}'(u)$ is tightly concentrated around $\text{depth}(u)$ using McDiarmid's inequality (5.3.13). Let X_i, for $i = 2 \ldots \text{depth}\, u$, be the choice made by u's depth-i ancestor. Then changing one X_i changes $\text{depth}'(u)$ by at most 1. So we get

$$\Pr\left[\text{depth}'(u) \geq \mathrm{E}\left[\text{depth}'(u)+t\right]\right] \leq e^{-2t^2/(\text{depth}(u)-1)} \tag{E.3.2}$$

and similarly

$$\Pr\left[\text{depth}'(u) \leq \mathrm{E}\left[\text{depth}'(u)-t\right]\right] \leq e^{-2t^2/(\text{depth}(u)-1)}. \tag{E.3.3}$$

Let $t = \sqrt{\frac{1}{2}D\ln(1/\epsilon)}$. Then the right-hand side of (E.3.2) and (E.3.3) becomes $e^{-D\ln(1/\epsilon)/(\text{depth}(u)-1)} < e^{-\ln(1/\epsilon)} = \epsilon$. For $\epsilon = \frac{1}{2}n^{-c-1}$, we get $t = \sqrt{\frac{1}{2}D\ln\left(2n^{c+1}\right)} = \sqrt{\frac{c+1}{2}D(\ln n+\ln 2)} = O\left(\sqrt{D\log n}\right)$ when c is constant.

For the lower bound on D', when can apply (E.3.3) to some single node u with $\text{depth}\,(u) = D$; this node by itself will give $D' \geq \frac{2}{3}D - O\left(\sqrt{D\log n}\right)$ with

[4] A less direct but still effective approach is to guess that $S(k)$ grows linearly, and find a and b such that $S(k) \leq ak+b$. For this we need $ak+b \leq 1+\frac{1}{2}(a(k-1)+b)+\frac{1}{2}(a(k-2)+b)$. The b's cancel, leaving $ak \leq 1+ak-\frac{3}{2}a$. Now the ak's cancel, leaving us with $0 \leq 1-\frac{3}{2}a$ or $a \geq 2/3$. We then go back and make $b = 1/3$ to get the right bound on $S(1)$, giving the bound $S(k) \leq \frac{2}{3}\cdot k+\frac{1}{3}$. We can then repeat the argument for $S(k) \geq a'k+b'$ to get a full bound $\frac{2}{3}k \leq S(k) \leq \frac{2}{3}k+\frac{1}{3}$.

probability at least $1-\frac{1}{2}n^{-c-1}$. For the upper bound, we need to take the maximum over all nodes. In general, an upper bound on the maximum of a bunch of random variables is likely to be larger than an upper bound on any one of the random variables individually, because there is a lot of room for one of the variables to get unlucky, but we can apply the union bound to get around this. For each individual u, we have $\Pr\left[\text{depth}'(u) \geq \frac{2}{3}D + O\left(\sqrt{D\log n}\right)\right] \leq \frac{1}{2}n^{-c-1}$, so $\Pr\left[D' \geq \frac{2}{3}D + O\left(\sqrt{D\log n}\right)\right] \leq \sum_u \frac{1}{2}n^{-c-1} = \frac{1}{2}n^{-c}$. This completes the proof.

E.3.2 Part testing

You are running a factory that produces parts for some important piece of machinery. Many of these parts are defective, and must be discarded. There are two levels of tests that can be performed:

- A normal test, which passes a part with probability 2/3.
- A rigorous test, which passes a part with probability 1/3.

At each step, the part inspectors apply the following rules to decide which test to apply:

- For the first part, a fair coin-flip decides between the tests.
- For subsequent parts, if the previous part passed, the inspectors become suspicious and apply the rigorous test; if it failed, they relax and apply the normal test.

For example, writing `N+` for a part that passes the normal test, `N-` for one that fails the normal test, `R+` for a part that passes the rigorous test, and `R-` for one that fails the rigorous test, a typical execution of the testing procedure might look like `N- N+ R- N- N+ R- N+ R- N- N- N- N+ R+ R- N+ R+`. This execution tests 16 parts and passes 7 of them.

Suppose that we test n parts. Let S be the number that pass.

1. Compute $\mathrm{E}[S]$.
2. Show that there a constant $c > 0$ such that, for any $t > 0$,

$$\Pr[|S - \mathrm{E}[S]| \geq t] \leq 2e^{-ct^2/n}. \tag{E.3.4}$$

Solution

Using McDiarmid's inequality and some cleverness Let X_i be the indicator variable for the event that part i passes, so that $S = \sum_{i=1}^{n} X_i$.

1. We can show by induction that $\mathrm{E}[X_i] = 1/2$ for all i. The base case is X_1, where $\Pr[\text{part 1 passes}] = \frac{1}{2}\Pr[\text{part 1 passes rigorous test}] + \frac{1}{2}\Pr[\text{part 1 passes normal test}] = \frac{1}{2}\left(\frac{1}{3} + \frac{2}{3}\right) = \frac{1}{2}$. For $i > 1$, $\mathrm{E}[X_{i-1}] = 1/2$ implies that part i is tested with the normal and rigorous tests with equal probability, so the analysis for X_1 carries through and gives $\mathrm{E}[X_i] = 1/2$ as well. Summing over all X_i gives $\mathrm{E}[S] = n/2$.

2. We can't use Chernoff, Hoeffding, or Azuma here, because the X_i are not independent, and do not form a martingale difference sequence even after centralizing them by subtracting off their expectations. So we are left with McDiarmid's inequality unless we want to do something clever and new (we don't). Applying McDiarmid to the X_i directly doesn't work so well, but we can make it work with a different set of variables that generate the same outcomes.

   Let $Y_i \in \{A, B, C\}$ be the grade of part i, where A means that it passes both the rigorous and the normal test, B means that fails the rigorous test but passes the normal test, and C means that it fails both tests. In terms of the X_i, $Y_i = A$ means $X_i = 1$, $Y_i = C$ means $X_i = 0$, and $Y_i = B$ means $X_i = 1 - X_{i-1}$ (when $i > 1$). We get the right probabilities for passing each test by assigning equal probabilities.

   We can either handle the coin-flip at the beginning by including an extra variable Y_0, or we can combine the coin-flip with Y_1 by assuming that Y_1 is either A or C with equal probability. The latter approach improves our bound a little bit since then we only have n variables and not $n + 1$.

   Now suppose that we fix all Y_j for $j \neq i$ and ask what happens if Y_i changes.

   (a) If $j < i$, then X_j is not affected by Y_i.
   (b) Let $k > i$ be such that $Y_k \in \{A, C\}$. Then X_k is not affected by Y_i, and neither is X_j for any $j > k$.

   It follows that changing Y_i can only change $X_i, \ldots, X_{i+\ell}$, where ℓ is the number of B grades that follow position i.

   There are two cases for the sequence $X_0 \ldots X_\ell$:

(a) If $X_i = 0$, then $X_1 = 1$, $X_2 = 0$, etc.

(b) If $X_i = 1$, then $X_1 = 0$, $X_2 = 1$, etc.

If ℓ is odd, changing Y_i thus has no effect on $\sum_{j=i}^{i+\ell} X_i$, while if ℓ is even, changing Y_i changes the sum by 1. In either case, the effect of changing Y_i is bounded by 1, and McDiarmid's inequality applies with $c_i = 1$, giving

$$\Pr\left[|S - \mathrm{E}\left[S\right]| \geq t\right] \leq 2e^{-2t^2/n}.$$

E.4 Assignment 4: due Wednesday, 2014-10-29, at 17:00

E.4.1 A doubling strategy

A common strategy for keeping the load factor of a hash table down is to double its size whenever it gets too full. Suppose that we start with a hash table of size 1 and double its size whenever two items in a row hash to the same location.

Effectively, this means that we attempt to insert all elements into a table of size 1; if two consecutive items hash to the same location, we start over and try to do the same to a table of size 2, and in general move to a table of size 2^{k+1} whenever any two consecutive elements hash to the same location in our table of size 2^k.

Assume that we are using an independently chosen 2-universal hash function for each table size. Show that the expected final table size is $O(n)$.

Solution

Let X be the random variable representing the final table size. Our goal is to bound $\mathrm{E}\left[X\right]$.

First let's look at the probability that we get at least one collision between consecutive elements when inserting n elements into a table with m locations. Because the pairs of consecutive elements overlap, computing the exact probability that we get a collision is complicated, but we only need an upper bound.

We have $n - 1$ consecutive pairs, and each produces a collision with probability at most $1/m$. This gives a total probability of a collision of at most $(n-1)/m$.

Let $k = \lceil \lg n \rceil$, so that $2^k \geq n$. Then the probability of a consecutive collision in a table with $2^{k+\ell}$ locations is at most $(n-1)/2^{k+\ell} < 2^k/2^{k+\ell} = 2^{-\ell}$.

Since the events that collisions occur at each table size are independent, we can compute, for $\ell > 0$,

$$\begin{aligned}\Pr\left[X = 2^{k+\ell}\right] &\le \Pr\left[X \ge 2^{k+\ell}\right] \\ &\le \prod_{i=0}^{\ell-1} 2^{-i} \\ &= 2^{-\ell(\ell-1)/2}.\end{aligned}$$

From this it follows that

$$\begin{aligned}\mathrm{E}[X] &= \sum_{i=0}^{\infty} 2^i \Pr\left[X = 2^i\right] \\ &< 2^k \Pr\left[X \le 2^k\right] + \sum_{l=1}^{\infty} 2^{k+\ell} \Pr\left[X = 2^{k+\ell}\right] \\ &\le 2^k + \sum_{l=1}^{\infty} 2^{k+\ell} \cdot 2^{-\ell(\ell-1)/2} \\ &\le 2^k + 2^k \sum_{l=1}^{\infty} 2^{\ell-\ell(\ell-1)/2} \\ &\le 2^k + 2^k \sum_{l=1}^{\infty} 2^{-(\ell^2-2\ell)/2} \\ &= O(2^k),\end{aligned}$$

since the series converges to some constant that does not depend on k. But we chose k so that $2^k = O(n)$, so this gives us our desired bound.

E.4.2 Hash treaps

Consider the following variant on a treap: instead of choosing the heap keys randomly, we choose a hash function $h : U \to [1 \dots m]$ from some strongly 2-universal hash family, and use $h(x)$ as the heap key for tree key x. Otherwise the treap operates as usual.[5]

Suppose that $|U| \ge n$. Show that there is a sequence of n distinct tree keys such that the total expected time to insert them into an initially empty hash treap is $\Omega(n^2/m)$.

[5] If m is small, we may get collisions in the heap key values. Assume in this case that a node will stop rising if its parent has the same key.

Solution

Insert the sequence $1 \dots n$.

Let us first argue by induction on i that all elements x with $h(x) = m$ appear as the uppermost elements of the right spine of the treap. Suppose that this holds for $i - 1$. If $h(i) = m$, then after insertion i is rotated up until it has a parent that also has heap key m; this extends the sequence of elements with heap key m in the spine by 1. Alternatively, if $h(i) < m$, then i is never rotated above an element x with $h(x) = m$, so the sequence of elements with heap key m is unaffected.

Because each new element has a larger tree key than all previous elements, inserting a new element i requires moving past any elements in the right spine, and in particular requires moving past any elements $j < i$ with $h(j) = m$. So the expected cost of inserting i is at least the expected number of such elements j. Because h is chosen from a strongly 2-universal hash family, $\Pr[h(j) = m] = 1/m$ for any j, and by linearity of expectation, $\mathrm{E}[|\{j < i \mid h(j) = m\}|] = (i - 1)/m$. Summing this quantity over all i gives a total expected insertion cost of at least $n(n-1)/2m = \Omega(n^2/m)$.

E.5 Assignment 5: due Wednesday, 2014-11-12, at 17:00

E.5.1 Agreement on a ring

Consider a ring of n processes at positions $0, 1, \dots n - 1$. Each process i has a value $A[i]$ that is initially 0 or 1. At each step, we choose a process r uniformly at random, and copy $A[r]$ to $A[(r + 1) \bmod n]$. Eventually, all $A[i]$ will have the same value.

Let $A_t[i]$ be the value of $A[i]$ at time t, and let $X_t = \sum_{i=0}^{n-1} A_t[i]$ be the total number of ones in A_t. Let τ be the first time at which $X_\tau \in \{0, n\}$.

1. Suppose that we start with $X_0 = k$. What is the probability that we eventually reach a state that is all ones?

2. What is $\mathrm{E}[\tau]$, assuming we start from the worst possible initial state?

Solution

This is a job for the optional stopping theorem. Essentially we are going to follow the same analysis from §9.4.1 for a random walk with two absorbing barriers, applied to X_t.

Let $\mathcal{F}_t$ be the σ-algebra generated by $A_0, \ldots, A_t$. Then $\{\mathcal{F}_t\}$ forms a filtration, and each X_t is $\mathcal{F}_t$-measurable.

1. For the first part, we show that $(X_t, \mathcal{F}_t)$ is a martingale. The intuition is that however the bits in A_t are arranged, there are always exactly the same number of positions where a zero can be replaced by a one as there are where a one can be replaced by a zero.

   Let R_t be the random location chosen in state A_t. Observe that

   $$\mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t, R_t = i\right] = X_t + A[i] - A[(i+1) \bmod n].$$

   But then

   $$\begin{aligned}\mathrm{E}\left[X_{t+1} \mid \mathcal{F}_t\right] &= \sum_{i=0}^{n-1} \frac{1}{n}\left(X_t + A[i] - A[(i+1) \bmod n]\right) \\ &= X_t + \frac{1}{n}X_t - \frac{1}{n}X_t \\ &= X_t,\end{aligned}$$

   which establishes the martingale property.

   We also have that (a) τ is a stopping time with respect to the $\mathcal{F}_i$; (b) $\Pr\left[\tau < \infty\right] = 1$, because from any state there is a nonzero chance that all $A[i]$ are equal n steps later; and (c) X_t is bounded between 0 and n. So the optional stopping theorem applies, giving

   $$\mathrm{E}\left[X_\tau\right] = \mathrm{E}\left[X_0\right] = k.$$

   But then

   $$\mathrm{E}\left[X_\tau\right] = n \cdot \Pr\left[X_\tau = n\right] + 0 \cdot \Pr\left[X_\tau = 0\right] = k,$$

   so $\mathrm{E}\left[X_\tau\right] = k/n$.

2. For the second part, we use a variant on the $X_t^2 - t$ martingale.

   Let Y_t count the number of positions i for which $A_t[i] = 1$ and $A_t[(i+1) \bmod n] = 0$. Then, conditioning on $\mathcal{F}_t$, we have

   $$X_{t+1}^2 = \begin{cases} X_t^2 + 2X_t + 1 & \text{with probability } Y_t/n, \\ X_t^2 - 2X_t + 1 & \text{with probability } Y_t/n, \text{ and} \\ X_t^2 & \text{otherwise.} \end{cases}$$

The conditional expectation sums to

$$\mathrm{E}\left[X_{t+1}^2 \;\middle|\; \mathcal{F}_t\right] = X_t^2 + 2Y_t/n.$$

Let $Z_t = X_t^2 - 2t/n$ when $t \le \tau$ and $X_\tau^2 - 2\tau/n$ otherwise. Then, for $t < \tau$, we have

$$\begin{aligned}
\mathrm{E}\left[Z_{t+1} \mid \mathcal{F}_t\right] &= \mathrm{E}\left[X_{t+1}^2 - 2(t+1)/n \;\middle|\; \mathcal{F}_t\right] \\
&= X_t^2 + 2Y_t/n - 2(t+1)/n \\
&= X_t^2 - 2t/n + 2Y_t/n - 2/n \\
&= Z_t + 2Y_t/n - 2/n \\
&\le Z_t.
\end{aligned}$$

For $t \ge \tau$, $Z_{t+1} = Z_t$, so in either case Z_t has the submartingale property.

We have previously established that τ has bounded expectation, and it's easy to see that Z_t has bounded step size. So the optional stopping theorem applies to Z_t, and $\mathrm{E}\left[Z_0\right] \le \mathrm{E}\left[Z_\tau\right]$.

Let $X_0 = k$. Then $\mathrm{E}\left[Z_0\right] = k^2$, and $\mathrm{E}\left[Z_\tau\right] = (k/n) \cdot n^2 - 2\,\mathrm{E}\left[\tau\right]/n = kn - 2\,\mathrm{E}\left[\tau\right]/n$. But then

$$k^2 \le kn - 2\,\mathrm{E}\left[\tau\right]/n$$

which gives

$$\begin{aligned}
\mathrm{E}\left[\tau\right] &\le \frac{kn - k^2}{2/n} \\
&= \frac{kn^2 - k^2 n}{2}.
\end{aligned}$$

This is maximized at $k = \lfloor n/2 \rfloor$, giving

$$\mathrm{E}\left[\tau\right] \le \frac{\lfloor n/2 \rfloor \cdot n^2 - (\lfloor n/2 \rfloor)^2 \cdot n}{2}.$$

For even n, this is just $n^3/4 - n^3/8 = n^3/8$.

For odd n, this is $(n-1)n^2/4 - (n-1)^2 n/8 = n^3/8 - n/8$. So there is a slight win for odd n from not being able to start with an exact half-and-half split.

To show that this bound applies in the worst case, observe that if we start with have contiguous regions of k ones and $n-k$ zeros in A_t, then (a) $Y_t = 1$, and (b) the two-region property is preserved in A_{t+1}. In this case, for $t < \tau$ it holds that $\mathrm{E}[Z_{t+1} \mid \mathcal{F}_t] = Z_t + 2Y_t/n - 2/n = Z_t$, so Z_t is a martingale, and thus $\mathrm{E}[\tau] = \frac{kn^2-k^2n}{2}$. This shows that the initial state with $\lfloor n/2 \rfloor$ consecutive zeros and $\lceil n/2 \rceil$ consecutive ones (or vice versa) gives the claimed worst-case time.

E.5.2 Shuffling a two-dimensional array

A programmer working under tight time constraints needs to write a procedure to shuffle the elements of an $n \times n$ array, so that all $(n^2)!$ permutations are equally likely. Some quick Googling suggests that this can be reduced to shuffling a one-dimensional array, for which the programmer's favorite language provides a convenient library routine that runs in time linear in the size of the array. Unfortunately, the programmer doesn't read the next paragraph about converting the 2-d array to a 1-d array first, and instead decides to pick one of the $2n$ rows or columns uniformly at random at each step, and call the 1-d shuffler on this row or column.

Let A^t be the state of the array after t steps, where each step shuffles one row or column, and let B be a random variable over permutations of the original array state that has a uniform distribution. Show that the 2-d shuffling procedure above is asympotically worse than the direct approach, by showing that there is some $f(n) = \omega(n)$ such that after $f(n)$ steps, $d_{TV}(A^t, B) = 1 - o(1)$.[6]

Solution

Consider the n diagonal elements in positions A_{ii}. For each such element, there is a $1/n$ chance at each step that its row or column is chosen. The

[6] I've been getting some questions about what this means, so here is an attempt to translate it into English.

Recall that $f(n)$ is $\omega(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ goes to infinity, and $f(n)$ is $o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ goes to zero.

The problem is asking you to show that there is some $f(n)$ that is more than a constant times n, such that the total variation distance between $A^{f(n)}$ becomes arbitrarily close to 1 for sufficiently large n.

So for example, if you showed that at $t = n^4$, $d_{TV}(A^t, B) \geq 1 - \frac{1}{\log^2 n}$, that would demonstrate the claim, because $\lim_{n\to\infty} \frac{n^2}{n}$ goes to infinity and $\lim_{n\to\infty} \frac{1/\log n}{1} = 0$. These functions are, of course, for illustration only. The actual process might or might not converge by time n^4.)

time until every diagonal node is picked at least once maps to the coupon collector problem, which means that it is $\Omega(n \log n)$ with high probability using standard concentration bounds.

Let C be the event that there is at least one diagonal element that is in its original position. If there is some diagonal node that has not moved, C holds; so with probability $1 - o(1)$, A^t holds at some time t that is $\Theta(n \log n) = \omega(n)$. But by the union bound, C holds in B with probability at most $n \cdot n^{-2} = 1/n$. So the difference between the probability of C in A^t and B is at least $1 - o(1) - 1/n = 1 - o(1)$.

E.6 Assignment 6: due Wednesday, 2014-12-03, at 17:00

E.6.1 Sampling colorings on a cycle

Devise a Las Vegas algorithm for sampling 3-colorings of a cycle.

Given $n > 1$ nodes in a cycle, your algorithm should return a random coloring X of the nodes with the usual constraint that no edge should have the same color on both endpoints. Your algorithm should generate all possible colorings with equal probability, and run in time polynomial in n on average.

Solution

Since it's a Las Vegas algorithm, Markov chain Monte Carlo is not going to help us here. So we can set aside couplings and conductance and just go straight for generating a solution.

First, let's show that we can generate uniform colorings of an n-node line in linear time. Let X_i be the color of node i, where $0 \le i < n$. Choose X_0 uniformly at random from all three colors; then for each $i > 0$, choose X_i uniformly at random from the two colors not chosen for X_{i-1}. Given a coloring, we can show by induction on i that it can be generated by this process, and because each choice is uniform and each coloring is generated only once, we get all $3 \cdot 2^{n-1}$ colorings of the line with equal probability.

Now we try hooking X_{n-1} to X_0. If $X_{n-1} = X_0$, then we don't have a cycle coloring, and have to start over. The probability that this event occurs is at most $1/2$, because for every path coloring with $X_{n-1} = X_0$, there is another coloring where we replace X_{n-1} with a color not equal to X_{n-2} or X_0. So after at most 2 attempts on average we get a good cycle coloring. This gives a total expected cost of $O(n)$.

E.6.2 A hedging problem

Suppose that you own a portfolio of n investment vehicles numbered $1, \ldots n$, where investment i pays out $a_{ij} \in \{-1, +1\}$ at time j, where $0 < j \leq m$. You have carefully chosen these investments so that your total payout $\sum_{i=1}^{n} a_{ij}$ for any time j is zero, eliminating all risk.

Unfortunately, in doing so you have run afoul of securities regulators, who demand that you sell off half of your holdings—a demand that, fortunately, you anticipated by making n even.

This will leave you with a subset S consisting of $n/2$ of the a_i, and your net worth at time t will now be $w_t = \sum_{j=1}^{t} \sum_{i \in S} a_{ij}$. If your net worth drops too low at any time t, all your worldly goods will be repossessed, and you will have nothing but your talent for randomized algorithms to fall back on.

Note: an earlier version of this problem demanded a tighter bound.

Show that when n and m are sufficiently large, it is always possible to choose a subset S of size $n/2$ so that $w_t \geq -m\sqrt{n \ln nm}$ for all $0 < t \leq m$, and give an algorithm that finds such a subset in time polynomial in n and m on average.

Solution

Suppose that we flip a coin independently to choose whether to include each investment i. There are two bad things that can happen:

1. We lose too much at some time t from the investments the coin chooses.

2. We don't get exactly $n/2$ heads.

If we can show that the sum of the probabilities of these bad events is less than 1, we get the existence proof we need. If we can show that it is enough less than 1, we also get an algorithm, because we can test in time $O(nm)$ if a particular choice works.

Let X_i be the indicator variable for the event that we include investment i. Then

$$\begin{aligned} w_t &= \sum_{i=1}^{n} \left(X_i \sum_{j=1}^{t} a_{ij} \right) \\ &= \sum_{i=1}^{n} \left(\left(\frac{1}{2} + \left(X_i - \frac{1}{2} \right) \right) \sum_{j=1}^{t} a_{ij} \right) \\ &= \frac{1}{2} \sum_{j=1}^{t} \sum_{i=1}^{n} a_{ij} + \sum_{i=1}^{n} \left(X_i - \frac{1}{2} \right) \left(\sum_{i=1}^{t} a_{ij} \right) \\ &= \sum_{i=1}^{n} \left(X_i - \frac{1}{2} \right) \left(\sum_{i=1}^{t} a_{ij} \right). \end{aligned}$$

Because $\left| X_i - \frac{1}{2} \right|$ is always $\frac{1}{2}$, $\mathrm{E}\left[X_i - \frac{1}{2} \right] = 0$, and each a_{ij} is ± 1, each term in the outermost sum is a zero-mean random variable that satisfies $\left| \left(X_i - \frac{1}{2} \right) \sum_{j=1}^{t} a_{ij} \right| \le \frac{t}{2} \le \frac{m}{2}$. So Hoeffding's inequality says

$$\begin{aligned} \Pr\left[w_t - \mathrm{E}\left[w_t \right] < m\sqrt{n \ln nm} \right] &\le e^{-m^2 n \ln nm / (2n(m/2)^2)} \\ &= e^{-2 \ln nm} \\ &= (nm)^{-2}. \end{aligned}$$

Summing over all t, the probability that this bound is violated for any t is at most $\frac{1}{n^2 m}$.

For the second source of error, we have $\Pr\left[\sum_{i=1}^{n} X_i \ne n/2 \right] = 1 - \binom{n}{2}/2^n = 1 - \Theta(1/\sqrt{n})$. So the total probability that the random assignment fails is bounded by $1 - \Theta(1/\sqrt{n}) + 1/n$, giving a probability that it succeeds of at least $\Theta(1/\sqrt{n}) - 1/(n^2 m) = \Theta(\sqrt{n})$. It follows that generating and testing random assignments gives an assignment with the desired characteristics after $\Theta(\sqrt{n})$ trials on average, giving a total expected cost of $\Theta(n^{3/2} m)$.

E.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

E.7.1 Double records (20 points)

Let $A[1 \ldots n]$ be an array holding the integers $1 \ldots n$ in random order, with all $n!$ permutations of the elements of A equally likely.

Call $A[k]$, where $1 \le k \le n$, a **record** if $A[j] < A[k]$ for all $j < k$.

Call $A[k]$, where $2 \le k \le n$, a **double record** if both $A[k-1]$ and $A[k]$ are records.

Give an asymptotic (big-Θ) bound on the expected number of double records in A.

Solution

Suppose we condition on a particular set of k elements appearing in positions $A[1]$ through $A[k]$. By symmetry, all $k!$ permutations of these elements are equally likely. Putting the largest two elements in $A[k-1]$ and $A[k]$ leaves $(k-2)!$ choices for the remaining elements, giving a probability of a double record at k of exactly $\frac{(k-2)!}{k!} = \frac{1}{k(k-1)}$.

Applying linearity of expectation gives a total expected number of double records of

$$\begin{aligned}
\sum_{k=2}^{n} \frac{1}{k(k-1)} &\le \sum_{k=2}^{n} \frac{1}{(k-1)^2} \\
&\le \sum_{k=2}^{\infty} \frac{1}{(k-1)^2} \\
&= \sum_{k=1}^{\infty} \frac{1}{k^2} \\
&= \frac{\pi^2}{6} \\
&= O(1).
\end{aligned}$$

Since the expected number of double records is at least $1/2 = \Omega(1)$ for $n \ge 2$, this gives a tight asymptotic bound of $\Theta(1)$.

I liked seeing our old friend $\pi^2/6$ so much that I didn't notice an easier

exact bound, which several people supplied in their solutions:

$$\begin{aligned}\sum_{k=2}^{n} \frac{1}{k(k-1)} &= \sum_{k=2}^{n}\left(\frac{1}{k-1} - \frac{1}{k}\right) \\ &= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=2}^{n} \frac{1}{k} \\ &= 1 - \frac{1}{n}. \\ &= O(1).\end{aligned}$$

E.7.2 Hipster graphs (20 points)

Consider the problem of labeling the vertices of a 3-regular graph with labels from $\{0, 1\}$ to maximize the number of *happy* nodes, where a node is *happy* if its label is the opposite of the majority of its three neighbors.

Give a *deterministic* algorithm that takes a 3-regular graph as input and computes, in time polynomial in the size of the graph, a labeling that makes at least half of the nodes happy.

Solution

This problem produced the widest range of solutions, including several very clever deterministic algorithms. Here are some examples.

Using the method of conditional probabilities If we are allowed a randomized algorithm, it's easy to get at exactly half of the nodes on average: simply label each node independently and uniformly at random, observe that each node individually has a 1/2 chance at happiness, and apply linearity of expectation.

To turn this into a deterministic algorithm, we'll apply the method of conditional expectations. Start with an unlabeled graph. At each step, pick a node and assign it a label that maximizes the expected number of happy nodes conditioned on the labeling so far, and assuming that all later nodes are labeled independently and uniformly at random. We can compute this conditional expectation in linear time by computing the value for each node (there are only 3^4 possible partial labelings of the node and its immediate neighbors, so computing the expected happiness of any particular node can be done in constant time by table lookup; to compute the table, we just enumerate all possible assignments to the unlabeled nodes). So in $O(n^2)$

time we get a labeling in which the number of happy nodes is at least the $n/2$ expected happy nodes we started with.

With a bit of care, the cost can be reduced to linear: because each new labeled node only affects its own probability of happiness and those of its three neighbors, we can update the conditional expectation by just updating the values for those four nodes. This gives $O(1)$ cost per step or $O(n)$ total.

Using hill climbing The following algorithm is an adaptation of the solution of Rose Sloan, and demonstrates that it is in fact possible to make *all* of the nodes happy in linear time.

Start with an arbitrary labeling (say, all 0). At each step, choose an unhappy node and flip its label. This reduces the number of monochromatic edges by at least 1. Because we have only $3n/2$ edges, we can repeat this process at most $3n/2$ times before it terminates. But it only terminates when there are no unhappy nodes.

To implement this in linear time, maintain a queue of all unhappy nodes.

E.7.3 Storage allocation (20 points)

Suppose that you are implementing a stack in an array, and you need to figure out how much space to allocate for the array. At time 0, the size of the stack is $X_0 = 0$. At each subsequent time t, the user flips an independent fair coin to choose whether to push onto the stack ($X_t = X_{t-1} + 1$) or pop from the stack ($X_t = X_{t-1} - 1$). The exception is that when the stack is empty, the user always pushes.

You are told in advance that the stack will only be used for n time units. Your job is to choose a size s for the array so that it will overflow at most half the time: $\Pr[\max_t X_t > s] < 1/2$. As an asymptotic (big-$O$) function of n, what is the smallest value of s you can choose?

Solution

We need two ideas here. First, we'll show that $X_t - t^2$ is a martingale, despite the fact that X_t by itself isn't. Second, we'll use the idea of stopping X_t when it hits $s+1$, creating a new martingale $Y_t = X_{t\wedge\tau}^2 - (t \wedge \tau)$ where τ is the first time where $X_t = s$ (and $t \wedge \tau$ is shorthand for $\min(t, \tau)$). We can then apply Markov's inequality to $X_{n\wedge\tau}$.

To save time, we'll skip directly to showing that Y_t is a martingale. There are two cases:

1. If $t < \tau$, then

$$\begin{aligned}\mathrm{E}\left[Y_{t+1} \mid Y_t, t < \tau\right] &= \frac{1}{2}\Big((X_t+1)^2 - (t+1)\Big) + \frac{1}{2}\Big((X_t-1)^2 - (t+1)\Big) \\ &= X_t^2 + 1 - (t+1) \\ &= X_t^2 - t \\ &= Y_t.\end{aligned}$$

2. If $t \geq \tau$, then $Y_{t+1} = Y_t$, so $\mathrm{E}\left[Y_{t+1} \mid Y_t, t \geq \tau\right] = Y_t$.

In either case the martingale property holds.

It follows that $\mathrm{E}\left[Y_n\right] = \mathrm{E}\left[Y_0\right] = 0$, or $\mathrm{E}\left[X_{n\wedge\tau}^2 - n\right] = 0$, giving $\mathrm{E}\left[X_{n\wedge\tau}^2\right] = n$. Now apply Markov's inequality:

$$\begin{aligned}\Pr\left[\max X_t > s\right] &= \Pr\left[X_{n\wedge\tau} \geq s+1\right] \\ &= \Pr\left[X_{n\wedge\tau}^2 \geq (s+1)^2\right] \\ &\leq \frac{\mathrm{E}\left[X_{n\wedge\tau}^2\right]}{(s+1)^2} \\ &= \frac{n}{(\sqrt{2n}+1)^2} \\ &< \frac{n}{2n} \\ &= 1/2.\end{aligned}$$

So $s = \sqrt{2n} = O(\sqrt{n})$ is enough.

E.7.4 Fault detectors in a grid (20 points)

A processing plant for rendering discarded final exam problems harmless consists of n^2 processing units arranged in an $n \times n$ grid with coordinates (i, j) each in the range 1 through n. We would like to monitor these processing units to make sure that they are working correctly, and have access to a supply of monitors that will detect failures. Each monitor is placed at some position (i, j), and will detect failures at any of the five positions (i, j), $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, and $(i, j+1)$ that are within the bounds of the grid. This plus-shaped detection range is awkward enough that the engineer designing the system has given up on figuring out how to pack the detectors properly, and instead places a detector at each grid location with independent probability 1/4.

The engineer reasons that since a typical monitor covers 5 grid locations, using $n^2/4$ monitors on average should cover $(5/4)n^2$ locations, with the extra monitors adding a little bit of safety to deal with bad random choices. So few if any processing units should escape.

1. Compute the exact expected number of processing units that are not within range of any monitor, as a function of n. You may assume $n > 0$.

2. Show that for any fixed c, the actual number of unmonitored processing units is within $O(n\sqrt{\log n})$ of the expected number with probability at least $1 - n^{-c}$.

Solution

1. This part is actually more annoying, because we have to deal with nodes on the edges. There are three classes of nodes:

   (a) The four corner nodes. To be unmonitored, each corner node needs to have no monitor in its own location or either of the two adjacent locations. This event occurs with probability $(3/4)^3$.

   (b) The $4(n-2)$ edge nodes. These have three neighbors in the grid, so they are unmonitored with probability $(3/4)^4$.

   (c) The $(n-2)^2$ interior nodes. These are each unmonitored with probability $(3/4)^5$.

   Adding these cases up gives a total expected number of unmonitored nodes of

   $$4 \cdot (3/4)^3 + 4(n-2) \cdot (3/4)^4 + (n-2)^2 \cdot (3/4)^5 = \frac{243}{1024}n^2 - \frac{81}{32}n + \frac{189}{64}. \tag{E.7.1}$$

   For $n = 1$, this analysis breaks down; instead, we can calculate directly that the sole node in the grid has a 3/4 chance of being unmonitored.

   Leaving the left-hand side of (E.7.1) in the original form is probably a good idea for understanding how the result works, but the right-hand side demonstrates that this strategy leaves slightly less than a quarter of the processing units uncovered on average.

2. Let X_{ij} be the indicator for a monitor at position (i, j). Recall that we have assumed that these variables are independent.

Let $S = f(X_{11}, \ldots, X_{nn})$ compute the number of uncovered processing units. Then changing a single X_{ij} changes the value of f by at most 5. So we can apply McDiarmid's inequality to get

$$\Pr\left[|S - \mathrm{E}\left[S\right]| \geq t\right] \leq 2e^{-2t^2/\left(\sum c_{ij}^2\right)} \quad = 2e^{-2t^2/\left(n^2 \cdot 5^2\right)} = 2e^{-2t^2/(25n^2)}.$$

(The actual bound is slightly better, because we are overestimating c_{ij} for boundary nodes.)

Set this to n^{-c} and solve for $t = n\sqrt{(25/2)(c \ln n + \ln 2)} = O(n\sqrt{\log n})$. Then the bound becomes $2e^{-c \ln n - \ln 2} = n^{-c}$, as desired.

Appendix F

Sample assignments from Spring 2013

F.1 Assignment 1: due Wednesday, 2013-01-30, at 17:00

F.1.1 Bureaucratic part

Send me email! My address is james.aspnes@gmail.com.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

F.1.2 Balls in bins

Throw m balls independently and uniformly at random into n bins labeled $1 \ldots n$. What is the expected number of positions $i < n$ such that bin i and bin $i + 1$ are both empty?

Solution

If we can figure out the probability that bins i and $i+1$ are both empty for some particular i, then by symmetry and linearity of expectation we can just multiply by $n-1$ to get the full answer.

For bins i and $i+1$ to be empty, every ball must choose another bin. This occurs with probability $(1-2/n)^m$. The full answer is thus $n(1-2/n)^m$, or approximately $ne^{-2m/n}$ when n is large.

F.1.3 A labeled graph

Suppose you are given a graph $G = (V, E)$, where $|V| = n$, and you want to assign labels to each vertex such that the sum of the labels of each vertex and its neighbors modulo $n+1$ is nonzero. Consider the naïve randomized algorithm that assigns a label in the range $0 \ldots n$ to each vertex independently and uniformly at random and then tries again if the resulting labeling doesn't work. Show that this algorithm will find a correct labeling in time polynomial in n on average.

Solution

We'll use the law of total probability. First observe that the probability that a random labeling yields a zero sum for any single vertex and its neighbors is exactly $1/(n+1)$; the easiest way to see this is that after conditioning on the values of the neighbors, there is only one value in $n+1$ that can be assigned to the vertex itself to cause a failure. Now sum this probability over all n vertices to get a probability of failure of at most $n/(n+1)$. It follows that after $n+1$ attempts on average (each of which takes $O(n^2)$ time to check all the neighborhood sums), the algorithm will find a good labeling, giving a total expected time of $O(n^3)$.

F.1.4 Negative progress

An algorithm has the property that if it has already run for n steps, it runs for an additional $n+1$ steps on average. Formally, let $T \geq 0$ be the random variable representing the running time of the algorithm, then

$$\mathrm{E}\left[T \mid T \geq n\right] = 2n + 1. \tag{F.1.1}$$

For each $n \geq 0$, what is the conditional probability $\Pr\left[T = n \mid T \geq n\right]$ that the algorithm stops just after its n-th step?

Solution

Expand (F.1.1) using the definition of conditional expectation to get

$$\begin{aligned} 2n+1 &= \sum_{x=0}^{\infty} x \Pr[T = x \mid T \geq n] \\ &= \sum_{x=0}^{\infty} x \frac{\Pr[T = x \wedge T \geq n]}{\Pr[T \geq n]} \\ &= \frac{1}{\Pr[T \geq n]} \sum_{x=n}^{\infty} x \Pr[T = x], \end{aligned}$$

which we can rearrange to get

$$\sum_{x=n}^{\infty} x \Pr[T = x] = (2n+1) \Pr[T \geq n], \tag{F.1.2}$$

provided $\Pr[T \geq n]$ is nonzero. We can justify this assumption by observing that (a) it holds for $n = 0$, because $T \geq 0$ always; and (b) if there is some $n > 0$ such that $\Pr[T \geq n] = 0$, then $\mathrm{E}[T \mid T \geq n-1] = n-1$, contradicting (F.1.1).

Substituting $n+1$ into (F.1.2) and subtracting from the original gives the equation

$$\begin{aligned} n \Pr[T = n] &= \sum_{x=n}^{\infty} x \Pr[T = x] - \sum_{x=n+1}^{\infty} x \Pr[T = x] \\ &= (2n+1) \Pr[T \geq n] - (2n+3) \Pr[T \geq n+1] \\ &= (2n+1) \Pr[T = n] + (2n+1) \Pr[T \geq n+1] - (2n+3) \Pr[T \geq n+1] \\ &= (2n+1) \Pr[T = n] - 2 \Pr[T \geq n+1]. \end{aligned}$$

Since we are looking for $\Pr[T = n \mid T \geq n] = \Pr[T = n] / \Pr[T \geq n]$, having an equation involving $\Pr[T \geq n+1]$ is a bit annoying. But we can borrow a bit of $\Pr[T = n]$ from the other terms to make it work:

$$\begin{aligned} n \Pr[T = n] &= (2n+1) \Pr[T = n] - 2 \Pr[T \geq n+1] \\ &= (2n+3) \Pr[T = n] - 2 \Pr[T = n] - 2 \Pr[T \geq n+1] \\ &= (2n+3) \Pr[T = n] - 2 \Pr[T \geq n]. \end{aligned}$$

A little bit of algebra turns this into

$$\Pr[T = n \mid T \geq n] = \frac{\Pr[T = n]}{\Pr[T \geq n]} = \frac{2}{n+3}.$$

F.2 Assignment 2: due Thursday, 2013-02-14, at 17:00

F.2.1 A local load-balancing algorithm

Suppose that we are trying to balance n jobs evenly between two machines. Job 1 chooses the left or right machine with equal probability. For $i > 1$, job i chooses the same machine as job $i-1$ with probability p, and chooses the other machine with probability $1-p$. This process continues until every job chooses a machine. We want to estimate how imbalanced the jobs are at the end of this process.

Let X_i be $+1$ if the i-th job chooses the left machine and -1 if it chooses the right. Then $S = \sum_{i=1}^{n} X_i$ is the difference between the number of jobs that choose the left machine and the number that choose the right. By symmetry, the expectation of S is zero. What is the variance of S as a function of p and n?

Solution

To compute the variance, we'll use (5.1.5), which says that $\operatorname{Var}\left[\sum_i X_i\right] = \sum_i \operatorname{Var}\left[X_i\right] + 2\sum_{i<j} \operatorname{Cov}\left[X_i, X_j\right]$.

Recall that $\operatorname{Cov}\left[X_i, X_j\right] = \operatorname{E}\left[X_i X_j\right] - \operatorname{E}\left[X_i\right]\operatorname{E}\left[X_j\right]$. Since the last term is 0 (symmetry again), we just need to figure out $\operatorname{E}\left[X_i X_j\right]$ for all $i \le j$ (the $i = j$ case gets us $\operatorname{Var}\left[X_i\right]$).

First, let's compute $\operatorname{E}\left[X_j = 1 \mid X_i = 1\right]$. It's easiest to do this starting with the $j = i$ case: $\operatorname{E}\left[X_i \mid X_i = 1\right] = 1$. For larger j, compute

$$\begin{aligned}\operatorname{E}\left[X_j \mid X_{j-1}\right] &= pX_{j-1} + (1-p)(-X_{j-1}) \\ &= (2p-1)X_{j-1}.\end{aligned}$$

It follows that

$$\begin{aligned}\operatorname{E}\left[X_j \mid X_i = 1\right] &= \operatorname{E}\left[(2p-1)X_{j-1} \mid X_i = 1\right] \\ &= (2p-1)\operatorname{E}\left[X_{j-1} \mid X_i = 1\right]. \qquad \text{(F.2.1)}\end{aligned}$$

The solution to this recurrence is $\operatorname{E}\left[X_j \mid X_i = 1\right] = (2p-1)^{j-i}$.

We next have

$$\begin{aligned}
\operatorname{Cov}[X_i, X_j] &= \operatorname{E}[X_i X_j] \\
&= \operatorname{E}[X_i X_j \mid X_i = 1] \Pr[X_i = 1] + \operatorname{E}[X_i X_j \mid X_i = -1] \Pr[X_i = -1] \\
&= \frac{1}{2} \operatorname{E}[X_j \mid X_i = 1] + \frac{1}{2} \operatorname{E}[-X_j \mid X_i = -1] \\
&= \frac{1}{2} \operatorname{E}[X_j \mid X_i = 1] + \frac{1}{2} \operatorname{E}[X_j \mid X_i = 1] \\
&= \operatorname{E}[X_j \mid X_i = 1] \\
&= (2p-1)^{j-i}
\end{aligned}$$

as calculated in (F.2.1).

So now we just need to evaluate the horrible sum.

$$\begin{aligned}
\sum_i \operatorname{Var}[X_i] + 2\sum_{i<j} \operatorname{Cov}[X_i, X_j] &= n + 2\sum_{i=1}^{n} \sum_{j=i+1}^{n} (2p-1)^{j-i} \\
&= n + 2\sum_{i=1}^{n} \sum_{k=1}^{n-i} (2p-1)^{k} \\
&= n + 2\sum_{i=1}^{n} \frac{(2p-1) - (2p-1)^{n-i+1}}{1-(2p-1)} \\
&= n + \frac{n(2p-1)}{1-p} - \frac{1}{1-p} \sum_{m=1}^{n} (2p-1)^{m} \\
&= n + \frac{n(2p-1)}{1-p} - \frac{(2p-1) - (2p-1)^{n+1}}{2(1-p)^2}.
\end{aligned} \tag{F.2.2}$$

This covers all but the $p = 1$ case, for which the geometric series formula fails. Here we can compute directly that $\operatorname{Var}[S] = n^2$, since S will be $\pm n$ with equal probability.

For smaller values of p, plotting (F.2.2) shows the variance increasing smoothly starting at 0 (for even n) or 1 (for odd n) at $p = 0$ to n^2 in the limit as p goes to 1, with an interesting intermediate case of n at $p = 1/2$, where all terms but the first vanish. This makes a certain intuitive sense: when $p = 0$, the processes alternative which machine they take, which gives an even split for even n and a discrepancy of ± 1 for odd n; when $p = 1/2$, the processes choose machines independently, giving variance n; and for $p = 1$, the processes all choose the same machine, giving n^2.

F.2.2 An assignment problem

Here is an algorithm for placing n balls into n bins. For each ball, we first select two bins uniformly at random without replacement. If at least one of the chosen bins is unoccupied, the ball is placed in the empty bin at no cost. If both chosen bins are occupied, we execute an expensive parallel scan operation to find an empty bin and place the ball there.

1. Compute the exact value of the expected number of scan operations.

2. Let $c > 0$. Show that the absolute value of the difference between the actual number of scan operations and the expected number is at most $O(\sqrt{cn \log n})$ with probability at least $1 - n^{-c}$.

Solution

1. Number the balls 1 to n. For ball i, there are $i-1$ bins already occupied, giving a probability of $\left(\frac{i-1}{n}\right)\left(\frac{i-2}{n-1}\right)$ that we choose an occupied bin on both attempts and incur a scan. Summing over all i gives us that the expected number of scans is:

$$\begin{aligned}\sum_{i=1}^{n}\left(\frac{i-1}{n}\right)\left(\frac{i-2}{n-1}\right) &= \frac{1}{n(n-1)}\sum_{i=1}^{n-1}(i^2-i)\\ &= \frac{1}{n(n-1)}\left(\frac{(n-1)n(2n-1)}{6} - \frac{(n-1)n)}{2}\right)\\ &= \frac{(2n-1)}{6} - \frac{1}{2}\\ &\frac{n-2}{3},\end{aligned}$$

   provided $n \geq 2$. For $n < 2$, we incur no scans.

2. It's tempting to go after this using Chernoff's inequality, but in this case Hoeffding gives a better bound. Let S be the number of scans. Then S is the sum of n independent Bernoulli random variables, so 5.3.4 says that $\Pr\left[|S - \mathrm{E}\left[S\right]| \geq t\right] \leq 2e^{-2t^2/n}$. Now let $t = \sqrt{cn \ln n} = O(\sqrt{cn \log n})$ to make the right-hand side $2n^{-2c} \leq n^{-c}$ for sufficiently large n.

F.2.3 Detecting excessive collusion

Suppose you have n students in some unnamed Ivy League university's *Introduction to Congress* class, and each generates a random ± 1 vote, with

both outcomes having equal probability. It is expected that members of the same final club will vote together, so it may be that many groups of up to k students each will all vote the same way (flipping a single coin to determine the vote of all students in the group, with the coins for different groups being independent). However, there may also be a much larger conspiracy of exactly m students who all vote the same way (again based on a single independent fair coin), in violation of academic honesty regulations.

Let $c > 0$. How large must m be in asymptotic terms as a function of n, k, and c so that the existence of a conspiracy can be detected solely by looking at the total vote, where the probability of error (either incorrectly claiming a conspiracy when none exists or incorrectly claiming no conspiracy when one exists) is at most n^{-c}?

Solution

Let S be the total vote. The intuition here is that if there is no conspiracy, S is concentrated around 0, and if there is a conspiracy, S is concentrated around $\pm m$. So if m is sufficiently large and $|S| \geq m/2$, we can reasonably guess that there is a conspiracy.

We need to prove two bounds: first, that the probability that we see $|S| \geq m/2$ when there is no conspiracy is small, and second, that the probability that we see $|S| < m/2$ when there is a conspiracy is large.

For the first case, let X_i be the total vote cast by the i-th group. This will be $\pm n_i$ with equal probability, where $n_i \leq k$ is the size of the group. This gives $\operatorname{E}[X_i] = 0$. We also have that $\sum n_i = n$.

Because the X_i are all bounded, we can use Hoeffding's inequality (5.3.2), so long as we can compute an upper bound on $\sum n_i^2$. Here we use the fact that $\sum n_i^2$ is maximized subject to $0 \leq n_i \leq k$ and $\sum n_i = 0$ by setting as many n_i as possible to k; this follows from convexity of $x \mapsto x^2$.[1] We thus have

$$\begin{aligned}\sum n_i^2 &\leq \lfloor n/k \rfloor k^2 + (n \bmod k)^2 \\ &\leq \lfloor n/k \rfloor k^2 + (n \bmod k)k \\ &\leq (n/k)k^2 \\ &= nk.\end{aligned}$$

[1]The easy way to see this is that if f is strictly convex, then f' is increasing. So if $0 < n_i \leq n_j < k$, increasing n_j by ϵ while decreasing n_i by ϵ leaves $\sum n_i$ unchanged while increasing $\sum f(n_i)$ by $\epsilon(f'(n_j) - f'(n_i)) + O(\epsilon^2)$, which will be positive when ϵ is small enough. So at any global maximum, we must have that at least one of n_i or n_j equals 0 or k for any $i \neq j$.

Now apply Hoeffding's inequality to get

$$\Pr\left[|S| \geq m/2\right] \leq 2e^{-(m/2)^2/2nk}.$$

We want to set m so that the right-hand side is less than n^{-c}. Taking logs as usual gives

$$\ln 2 - m^2/8nk \leq -c \ln n,$$

so the desired bound holds when

$$\begin{aligned} m &\geq \sqrt{8nk(c \ln n + \ln 2)} \\ &= \Omega(\sqrt{ckn \log n}). \end{aligned}$$

For the second case, repeat the above analysis on the $n - m$ votes except the $\pm m$ from the conspiracy. Again we get that if $m = \Omega(\sqrt{ckn \log n})$, the probability that these votes exceed $m/2$ is bounded by n^{-c}. So in both cases $m = \Omega(\sqrt{ckn \log n})$ is enough.

F.3 Assignment 3: due Wednesday, 2013-02-27, at 17:00

F.3.1 Going bowling

For your crimes, you are sentenced to play n frames of **bowling**, a game that involves knocking down pins with a heavy ball, which we are mostly interested in because of its complicated scoring system.

In each frame, your result may be any of the integers 0 through 9, a **spare** (marked as `/`), or a **strike** (marked as `X`). We can think of your result on the i-th frame as a random variable X_i. Each result gives a base score B_i, which is equal to X_i when $X_i \in \{0 \ldots 9\}$ and 10 when $X_i \in \{\texttt{/}, \texttt{X}\}$. The actual score Y_i for frame i is the sum of the base scores for the frame and up to two subsequent frames, according to the rule:

$$Y_i = \begin{cases} B_i & \text{when } X_i \in \{0 \ldots 9\}, \\ B_i + B_{i+1} & \text{when } X_i = \texttt{/}, \text{ and} \\ B_i + B_{i+1} + B_{i+2} & \text{when } X_i = \texttt{X}. \end{cases}$$

To ensure that B_{i+2} makes sense even for $i = n$, assume that there exist random variables X_{n+1} and X_{n+2} and the corresponding B_{n+1} and B_{n+2}.

Suppose that the X_i are independent (but not necessarily identically distributed). Show that your final score $S = \sum_{i=1}^{n} Y_i$ is exponentially concentrated[2] around its expected value.

Solution

This is a job for McDiarmid's inequality (5.3.13). Observe that S is a function of $X_1 \ldots X_{n+2}$. We need to show that changing any one of the X_i won't change this function by too much.

From the description of the Y_i, we have that X_i can affect any of Y_{i-2} (if $X_{i-2} = \texttt{X}$), Y_{i-1} (if $X_{i-1} = \texttt{/}$) and Y_i. We can get a crude bound by observing that each Y_i ranges from 0 to 30, so changing X_i can change $\sum Y_i$ by at most ± 90, giving $c_i \leq 90$. A better bound can be obtained by observing that X_i contributes only B_i to each of Y_{i-2} and Y_{i-1}, so changing X_i can only change these values by up to 10; this gives $c_i \leq 50$. An even more pedantic bound can be obtained by observing that X_1, X_2, X_{n+1}, and X_{n+2} are all special cases, with $c_1 = 30$, $c_2 = 40$, $c_{n+1} = 20$, and $c_{n+2} = 10$, respectively; these values can be obtained by detailed meditation on the rules above.

We thus have $\sum_{i=1}^{n+2} c_i^2 = (n-2)50^2 + 30^2 + 40^2 + 20^2 + 10^2 = 2500(n-2) + 3000 = 2500n - 2000$, assuming $n \geq 2$. This gives $\Pr[|S - \mathrm{E}[S]| \geq t] \leq \exp(-2t^2/(2500n - 2000))$, with the symmetric bound holding on the other side as well.

For the standard game of bowling, with $n = 10$, this bound starts to bite at $t = \sqrt{11500} \approx 107$, which is more than a third of the range between the minimum and maximum possible scores. There's a lot of variance in bowling, but this looks like a pretty loose bound for players who don't throw a lot of strikes. For large n, we get the usual bound of $O\left(\sqrt{n \log n}\right)$ with high probability: the averaging power of endless repetition eventually overcomes any slop in the constants.

F.3.2 Unbalanced treaps

Recall that a **treap** (§6.3) is only likely to be balanced if the sequence of insert and delete operations applied to it is independent of the priorities chosen by the algorithm.

Suppose that we insert the keys 1 through n into a treap with random priorities as usual, but then allow the adversary to selectively delete whichever keys it wants to after observing the priorities assigned to each key.

[2] This means "more concentrated than you can show using Chebyshev's inequality."

Show that there is an adversary strategy that produces a path in the treap after deletions that has expected length $\Omega(\sqrt{n})$.

Solution

An easy way to do this is to produce a tree that consists of a single path, which we can do by arranging that the remaining keys have priorities that are ordered the same as their key values.

Here's a simple strategy that works. Divide the keys into $\sqrt{n}$ ranges of $\sqrt{n}$ keys each ($1 \ldots \sqrt{n}$, $\sqrt{n}+1 \ldots 2\sqrt{n}$, etc.).[3] Rank the priorities from 1 to n. From each range $(i-1)\sqrt{n} \ldots i\sqrt{n}$, choose a key to keep whose priority is also ranked in the range $(i-1)\sqrt{n} \ldots i\sqrt{n}$ (if there is one), or choose no key (if there isn't). Delete all the other keys.

For a particular range, we are drawing $\sqrt{n}$ samples without replacement from the n priorities, and there are $\sqrt{n}$ possible choices that cause us to keep a key in that range. The probability that every draw misses is $\prod_{i=1}^{\sqrt{n}}(1-\sqrt{n}/(n-i+1)) \leq (1-1/\sqrt{n})^{\sqrt{n}} \leq e^{-1}$. So each range contributes at least $1-e^{-1}$ keys on average. Summing over all $\sqrt{n}$ ranges gives a sequence of keys with increasing priorities with expected length at least $(1-e^{-1})\sqrt{n} = \Omega(\sqrt{n})$.

An alternative solution is to apply the **Erdős-Szekeres theorem** [ES35], which says that every sequence of length k^2+1 has either an increasing subsequence of length $k+1$ or a decreasing sequence of $k+1$. Consider the sequence of priorities corresponding to the keys $1 \ldots n$; letting $k = \lfloor \sqrt{n-1} \rfloor$ gives a subsequence of length at least $\sqrt{n-1}$ that is either increasing or decreasing. If we delete all other elements of the treap, the elements corresponding to this subsequence will form a path, giving the desired bound. Note that this does not require any probabilistic reasoning at all.

Though not required for the problem, it's possible to show that $\Theta(\sqrt{n})$ is the best possible bound here. The idea is that the number of possible sequences of keys that correspond to a path of length k in a binary search tree is exactly $\binom{n}{k}2^{k-1}$; the $\binom{n}{k}$ corresponds to choosing the keys in the path, and the 2^{k-1} is because for each node except the last, it must contain either the smallest or the largest of the remaining keys because of the binary search tree property.

Since each such sequence will be a treap path only if the priorities are decreasing (with probability $1/k!$), the union bound says that the probability

[3] To make our life easer, we'll assume that n is a square. This doesn't affect the asymptotic result.

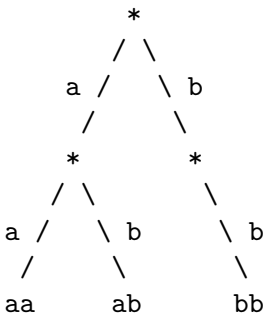


Figure F.1: A radix tree, storing the strings `aa`, `ab`, and `ba`.

of having any length-k paths is at most $\binom{n}{k}2^{k-1}/k!$. But

$$\begin{aligned}\binom{n}{k}2^{k-1}/k! &\leq \frac{(2n)^k}{2(k!)^2} \\ &\geq \frac{(2n)^k}{2(k/e)^{2k}} \\ &= \frac{1}{2}(2e^2n/k^2)^k.\end{aligned}$$

This is exponentially small for $k \gg \sqrt{2e^2n}$, showing that with high probability all possible paths have length $O(\sqrt{n})$.

F.3.3 Random radix trees

A **radix tree** over an alphabet of size m is a tree data structure where each node has up to m children, each corresponding to one of the letters in the alphabet. A string is represented by a node at the end of a path whose edges are labeled with the letters in the string in order. For example, in Figure F.1, the string `ab` is stored at the node reached by following the `a` edge out of the root, then the `b` edge out of this child.

The only nodes created in the radix tree are those corresponding to stored keys or ancestors of stored keys.

Suppose you have a radix tree into which you have already inserted n strings of length k from an alphabet of size m, generated uniformly at random with replacement. What is the expected number of new nodes you need to create to insert a new string of length k?

Solution

We need to create a new node for each prefix of the new string that is not already represented in the tree.

For a prefix of length ℓ, the chance that none of the n strings have this prefix is exactly $\left(1 - m^{-\ell}\right)^n$. Summing over all ℓ gives that the expected number of new nodes is $\sum_{\ell=0}^{k} \left(1 - m^{-\ell}\right)^n$.

There is no particularly clean expression for this, but we can observe that $(1 - m^{-\ell})^n \leq \exp(-nm^{-\ell})$ is close to zero for $\ell < \log_m n$ and close to 1 for $\ell > \log_m n$. This suggests that the expected value is $k - \log_m n + O(1)$.

F.4 Assignment 4: due Wednesday, 2013-03-27, at 17:00

F.4.1 Flajolet-Martin sketches with deletion

A **Flajolet-Martin sketch** [FNM85] is a streaming data structure for approximately counting the number of distinct items n in a large data stream using only $m = O(\log n)$ bits of storage.[4] The idea is to use a hash function h that generates each value $i \in \{1, \ldots m\}$ with probability 2^{-i}, and for each element x that arrives in the data stream, we write a 1 to $A[h(x)]$. (With probability 2^{-m} we get a value outside this range and write nowhere.) After inserting n distinct elements, we estimate n as $\hat{n} = 2^{\hat{k}}$, where $\hat{k} = \max\{k \mid A[k] = 1\}$, and argue that this is likely to be reasonably close to n.

Suppose that we modify the Flajolet-Martin sketch to allow an element x to be deleted by writing 0 to $A[h(x)]$. After n insertions and d deletions (of distinct elements in both cases), we estimate the number of remaining elements $n - d$ as before by $\widehat{n - d} = 2^{\hat{k}}$, where $\hat{k} = \max\{k \mid A[k] = 1\}$.

Assume that we never delete an element that has not previously been inserted, and that the values of h are for different inputs are independent of each other and of the sequence of insertions and deletions.

Show that there exist constants $c > 1$ and $\epsilon > 0$, such that for n sufficiently large, after inserting n distinct elements then deleting $d \leq \epsilon n$ of them, $\Pr\left[(n-d)/c \leq \widehat{n - d} \leq (n-d)c\right] \geq 2/3$.

[4] If you actually need to do this, there exist better data structures for this problem. See [KNW10].

Solution

We'll apply the usual error budget approach and show that the probability that $\widehat{n-d}$ is too big and the probability that $\widehat{n-d}$ is too small are both small. For the moment, we will leave c and ϵ as variables, and find values that work at the end.

Let's start with the too-big side. To get $A[k] = 1$, we need $h(x_i) = k$ for some x_i that is inserted but not subsequently deleted. There are $n-d$ such x_i, and each gives $h(x_i) = k$ with probability 2^{-k}. So $\Pr[A[k] = 1] \leq (n-d)2^{-k}$. This gives

$$\begin{aligned}
\Pr\left[\widehat{n-d} \geq (n-d)c\right] &= \Pr\left[\hat{k} \geq \lceil \lg((n-d)c)\rceil\right] \\
&\leq \sum_{k=\lceil \lg((n-d)c)\rceil}^{\infty} (n-d)2^{-k} \\
&= 2(n-d)2^{-\lceil \lg((n-d)c)\rceil} \\
&\leq \frac{2}{c}.
\end{aligned}$$

On the too-small side, fix $k = \lceil \lg((n-d)/c))\rceil$. Since $A[k] = 1$ gives $\hat{k} \geq k \geq \lceil \lg((n-d)/c))\rceil$, we have $\Pr\left[\widehat{n-d} < (n-d)/c\right] = \Pr\left[\hat{k} < \lg(n-d)/c\right] \leq \Pr[A[k] = 0]$. (We might be able to get a better bound by looking at larger indices, but to solve the problem this one k will turn out to be enough.)

Let $x_1 \ldots x_{n-d}$ be the values that are inserted and not later deleted, and $x_{n-d+1} \ldots x_n$ the values that are inserted and then deleted. For $A[k]$ to be zero, either (a) no x_i for i in $1 \ldots x_{n-d}$ has $h(x_i) = k$; or (b) some x_i for i in $n-d+1 \ldots x_n$ has $h(x_i) = k$. The probability of the first event is

$\left(1-2^{-k}\right)^{n-d}$; the probability of the second is $1-\left(1-2^{-k}\right)^{d}$. So we have

$$\begin{aligned}
\Pr\left[A[k]=0\right] &\leq \left(1-2^{-k}\right)^{n-d} + \left(1-\left(1-2^{-k}\right)^{d}\right) \\
&\leq \exp\left(-2^{-k}(n-d)\right) + \left(1-\exp\left(-\left(2^{-k}+2^{-2k}\right)d\right)\right) \\
&\leq \exp\left(-2^{-\lceil \lg((n-d)/c))\rceil}(n-d)\right) + \left(1-\exp\left(-2\cdot 2^{-\lceil \lg((n-d)/c))\rceil}d\right)\right) \\
&\leq \exp\left(-2^{-\lg((n-d)/c))}(n-d)\right) + \left(1-\exp\left(-2\cdot 2^{-\lg((n-d)/c))+1}d\right)\right) \\
&= e^{-c} + \left(1-\exp\left(-\frac{4cd}{n-d}\right)\right) \\
&\leq e^{-c} + \frac{4cd}{n-d} \\
&\leq e^{-c} + \frac{4c\epsilon}{1-\epsilon}.
\end{aligned}$$

So our total probability of error is bounded by $\frac{2}{c} + e^{-c} + \frac{4c\epsilon}{1-\epsilon}$. Let $c = 8$ and $\epsilon = 1/128$ to make this less than $\frac{1}{4} + e^{-8} + \frac{128}{127}\cdot\frac{1}{16} \approx 0.313328 < 1/3$, giving the desired bound.

F.4.2 An adaptive hash table

Suppose we want to build a hash table, but we don't know how many elements we are going to put in it, and because we allow undisciplined C programmers to obtain pointers directly to our hash table entries, we can't move an element once we assign it a position to it. Here we will consider a data structure that attempts to solve this problem.

Construct a sequence of tables $T_0, T_1, \ldots$, where each T_i has $m_i = 2^{2^i}$ slots. For each table T_i, choose k independent strongly 2-universal hash functions $h_{i1}, h_{i2}, \ldots h_{ik}$.

The insertion procedure is given in Algorithm F.1. The essentially idea is that we make k attempts (each with a different hash function) to fit x into T_0, then k attempts to fit it in T_1, and so on.

If the tables T_i are allocated only when needed, the space complexity of this data structure is given by the sum of m_i for all tables that have at least one element.

Show that for any fixed $\epsilon > 0$, there is a constant k such that after inserting n elements:

1. The expected cost of an additional insertion is $O(\log\log n)$, and
2. The expected space complexity is $O\left(n^{2+\epsilon}\right)$.

```
1 procedure insert(x)
2     for i ← 0 to ∞ do
3         for j ← 1 to k do
4             if T_i[h_ij(x)] = ⊥ then
5                 T_i[h_ij(x)] ← x
6                 return
```

Algorithm F.1: Adaptive hash table insertion

Solution

The idea is that we use T_{i+1} only if we get a collision in T_i. Let X_i be the indicator for the event that there is a collision in T_i. Then

$$\mathrm{E}\left[\text{steps}\right] \le 1 + \sum_{i=0}^{\infty} \mathrm{E}\left[X_i\right] \tag{F.4.1}$$

and

$$\mathrm{E}\left[\text{space}\right] \le m_0 + \sum_{i=0}^{\infty} \mathrm{E}\left[X_i\right] m_{i+1}. \tag{F.4.2}$$

To bound $\mathrm{E}\left[X_i\right]$, let's calculate an upper bound on the probability that a newly-inserted element x_{n+1} collides with any of the previous n elements $x_1 \ldots x_n$ in table T_i. This occurs if, for every location $h_{ij}(x_{n+1})$, there is some x_r and some j' such that $h_{ij'}(x_r) = h_{ij}(x_{n+1})$. The chance that this occurs for any particular j, j', and r is at most $1/m_i$ (if $j = j'$, use 2-universality of h_{ij}, and if $j \neq j'$, use independence and uniformity), giving a chance that it occurs for fixed j that is at most n/m_i. The chance that it occurs for all j is at most $(n/m_i)^k$, and the expected number of such collisions summed over any $n+1$ elements that we insert is bounded by $n(n/m_i)^k$ (the first element can't collide with any previous elements). So we have $\mathrm{E}\left[X_i\right] \le \min\left(1, n(n/m_i)^k\right)$.

Let ℓ be the largest value such that $m_\ell \le n^{2+\epsilon}$. We will show that, for an appropriate choice of k, we are sufficiently unlikely to get a collision in round ℓ that the right-hand sides of (F.4.2) and (F.4.2) end up being not much more than the corresponding sums up to $\ell - 1$.

From our choice of ℓ, it follows that (a) $\ell \le \lg\lg n^{2+\epsilon} = \lg\lg n + \lg(2+\epsilon) = O(\log\log n)$; and (b) $m_{\ell+1} > n^{2+\epsilon}$, giving $m_\ell = \sqrt{m_{\ell+1}} > n^{1+\epsilon/2}$. From this we get $\mathrm{E}\left[X_\ell\right] \le n(n/m_\ell)^k < n^{1-k\epsilon/2}$.

By choosing k large enough, we can make this an arbitrarily small polynomial in n. Our goal is to wipe out the $\mathrm{E}[X_\ell]\, m_{\ell+1}$ and subsequent terms in (F.4.2).

Observe that $\mathrm{E}[X_i]\, m_{i+1} \le n(n/m_i)^k m_i^2 = n^{k+1} m_i^{2-k}$. Let's choose k so that this is at most $1/m_i$, when $i \ge \ell$, so we get a nice convergent series.[5] This requires $n^{k+1} m_i^{3-k} \le 1$ or $k + 1 + (3-k)\log_n m_i \le 0$. If $i \ge \ell$, we have $\log_n m_i > 1 + \epsilon/2$, so we win if $k + 1 + (3-k)(1+\epsilon/2) \le 0$. Setting $k \ge 8/\epsilon + 3$ works. (Since we want k to be an integer, we probably want $k = \lceil 8/\epsilon \rceil + 3$ instead.)

So now we have

$$\begin{aligned}
\mathrm{E}[\text{space}] &\le m_0 + \sum_{i=0}^{\infty} \mathrm{E}[X_i]\, m_{i+1} \\
&\le \sum_{i=0}^{\ell} m_i + \sum_{i=\ell}^{\infty} \frac{1}{m_i} \\
&\le 2m_\ell + \frac{2}{m_\ell} \\
&= O(n^{2+\epsilon}).
\end{aligned}$$

For $\mathrm{E}[\text{steps}]$, compute the same sum without all the m_{i+1} factors. This makes the tail terms even smaller, so they is still bounded by a constant, and the head becomes just $\sum_{i=0}^{\ell} 1 = O(\log\log n)$.

F.4.3 An odd locality-sensitive hash function

A deranged computer scientist decides that if taking one bit from a random index in a bit vector is a good way to do locality-sensitive hashing (see §7.8.4), then taking the exclusive OR of k independently chosen indices must be even better.

Formally, given a bit-vector $x_1 x_2 \ldots x_n$, and a sequence of indices $i_1 i_2 \ldots i_k$, define $h_i(x) = \bigoplus_{j=1}^{k} x_{i_j}$. For example, if $x = 00101$ and $i = 3, 5, 2$, $h_i(x) = 1 \oplus 1 \oplus 0 = 0$.

Suppose x and y are bit-vectors of length n that differ in m places.

1. Give a *closed-form expression* for the probability that $h_i(x) \ne h_i(y)$, assuming i consists of k indices chosen uniformly and independently at random from $1 \ldots n$

[5] We could pick a smaller k, but why not make things easier for ourselves?

2. Use this to compute the exact probability that $h_i(x) \neq h_i(y)$ when $m = 0$, $m = n/2$, and $m = n$.

Hint: You may find it helpful to use the identity $(a \bmod 2) = \frac{1}{2}(1-(-1)^a)$.

Solution

1. Observe that $h_i(x) \neq h_i(y)$ if an only if i chooses an odd number of indices where x and y differ. Let $p = m/n$ be the probability that each index in i hits a position where x and y differ, and let $q = 1 - p$. Then the event that we get an odd number of differences is

$$\begin{aligned}
\sum_{j=0}^{k}(j \bmod 2)\binom{k}{j}p^j q^{k-j} &= \sum_{j=0}^{k}\frac{1}{2}\left(1-(-1)^j\right)\binom{k}{j}p^j q^{k-j} \\
&= \frac{1}{2}\sum_{j=0}^{k}\binom{k}{j}p^j q^{k-j} - \frac{1}{2}\sum_{j=0}^{k}\binom{k}{j}(-p)^j q^{k-j} \\
&= \frac{1}{2}(p+q)^k - \frac{1}{2}(-p+q)^k \\
&= \frac{1-(1-2(m/n))^k}{2}.
\end{aligned}$$

2.
   - For $m = 0$, this is $\frac{1-1^k}{2} = 0$.
   - For $m = n/2$, it's $\frac{1-0^k}{2} = \frac{1}{2}$ (assuming $k > 0$).
   - For $m = n$, it's $\frac{1-(-1)^k}{2} = (k \bmod 2)$.

In fact, the chances of not colliding as a function of m are symmetric around $m = n/2$ if k is even and increasing if k is odd. So we can only hope to use this as locality-sensitive hash function in the odd case.

F.5 Assignment 5: due Friday, 2013-04-12, at 17:00

F.5.1 Choosing a random direction

Consider the following algorithm for choosing a random direction in three dimensions. Start at the point $(X_0, Y_0, Z_0) = (0, 0, 0)$. At each step, pick one of the three coordinates uniformly at random and add ± 1 to it with equal probability. Continue until the resulting vector has length at least k, i.e., until $X_t^2 + Y_t^2 + Z_t^2 \geq k^2$. Return this vector.

What is the expected running time of this algorithm, as an asymptotic function of k?

Solution

The trick is to observe that $X_t^2 + Y_t^2 + Z_t^2 - t$ is a martingale, essentially following the same analysis as for $X_t^2 - t$ for a one-dimensional random walk. Suppose we pick X_t to change. Then

$$\begin{aligned}\mathrm{E}\left[X_{t+1}^2 \;\middle|\; X_t\right] &= \frac{1}{2}\left((X_t+1)^2 + (X_t-1)^2\right)\\ &= X_t^2 + 1.\end{aligned}$$

So

$$\mathrm{E}\left[X_{t+1}^2 + Y_{t+1}^2 + Z_{t+1}^2 - (t+1) \;\middle|\; X_t, Y_t, Z_t, X \text{ changes}\right] = X_t^2 + Y_t^2 + Z_t^2 - t.$$

But by symmetry, the same equation holds if we condition on Y or Z changing. It follows that $\mathrm{E}\left[X_{t+1}^2 + Y_{t+1}^2 + Z_{t+1}^2 - (t+1) \mid X_t, Y_t, Z_t\right] = X_t^2 + Y_t^2 + Z_t^2 - t$, and that we have a martingale as claimed.

Let τ be the first time at which $X_t^2 + Y_t^2 + Z_t^2 \le k^2$. From the optional stopping theorem (specifically, the bounded-increments case of Theorem 9.3.1), $\mathrm{E}\left[X_\tau^2 + Y_\tau^2 + Z_\tau^2 - \tau\right] = 0$, or equivalently $\mathrm{E}\left[\tau\right] = \mathrm{E}\left[X_\tau^2 + Y_\tau^2 + Z_\tau^2\right]$. This immediately gives $\mathrm{E}\left[\tau\right] \ge k^2$.

To get an upper bound, observe that $X_{\tau-1}^2 + Y_{\tau-1}^2 + Z_{\tau-1}^2 < k^2$, and that exactly one of these three term increases between $\tau - 1$ and τ. Suppose it's X (the other cases are symmetric). Increasing X by 1 sets $X_\tau^2 = X_{\tau-1}^2 + 2X_{\tau-1} + 1$. So we get

$$\begin{aligned}X_\tau^2 + Y_\tau^2 + Z_\tau^2 &= \left(X_{\tau-1}^2 + Y_{\tau-1}^2 + Z_{\tau-1}^2\right) + 2X_{\tau-1} + 1\\ &< k^2 + 2k + 1.\end{aligned}$$

So we have $k^2 \le \mathrm{E}\left[\tau\right] < k^2 + 2k + 1$, giving $\mathrm{E}\left[\tau\right] = \Theta(k^2)$. (Or $k^2 + O(k)$ if we are feeling really precise.)

F.5.2 Random walk on a tree

Consider the following random walk on a (possibly unbalanced) binary search tree: At each step, with probability 1/3 each, move to the current node's parent, left child, or right child. If the target node does not exist, stay put.

Suppose we adapt this random walk using Metropolis-Hastings (see §10.3.4) so that the probability of each node at depth d in the stationary distribution is proportional to α^{-d}.

Use a coupling argument to show that, for any constant $\alpha > 2$, this adapted random walk converges in $O(D)$ steps, where D is the depth of the tree.

Solution

As usual, let X_t be a copy of the chain starting in an arbitrary initial state and Y_t be a copy starting in the stationary distribution.

From the Metropolis-Hastings algorithm, the probability that the walk moves to a particular child is $1/3\alpha$, so the probability that the depth increases after one step is at most $2/3\alpha$. The probability that the walk moves to the parent (if we are not already at the root) is $1/3$.

We'll use the same choice (left, right, or parent) in both the X and Y processes, but it may be that only one of the particles moves (because the target node doesn't exist). To show convergence, we'll track $Z_t = \max(\text{depth}(X_t), \text{depth}(Y_t))$. When $Z_t = 0$, both X_t and Y_t are the root node.

There are two ways that Z_t can change:

1. Both processes choose "parent"; if Z_t is not already 0, $Z_{t+1} = Z_t - 1$. This case occurs with probability $1/3$.

2. Both processes choose one of the child directions. If the appropriate child exists for the deeper process (or for either process if they are at the same depth), we get $Z_{t+1} = Z_t + 1$. This even occurs with probability at most $2/3\alpha < 1/3$.

So the expected change in Z_{t+1} conditioned on $Z_t > 0$ is at most $-1/3 + 2/3\alpha = -(1/3)(2/\alpha - 1)$. Let τ be the first time at which $Z_t = 0$. Then the process $Z'_t = Z_t - (1/3)(2/\alpha - 1)t$ for $t \leq \tau$ and 0 for $t > \tau$ is a supermartingale, so $\mathrm{E}[Z_\tau] = \mathrm{E}[Z_0] = \mathrm{E}[\max(\text{depth}(X_0), \text{depth}(Y_0))] \leq D$. This gives $\mathrm{E}[\tau] \leq \frac{3D}{2/\alpha - 1}$.

F.5.3 Sampling from a tree

Suppose we want to sample from the stationary distribution of the Metropolis-Hastings walk in the previous problem, but we don't want to do an actual random walk. Assuming $\alpha > 2$ is a constant, give an algorithm for sampling *exactly* from the stationary distribution that runs in constant expected time.

Your algorithm should not require knowledge of the structure of the tree. Its only input should be a pointer to the root node.

Clarification added 2013-04-09: Your algorithm can determine the children of any node that it has already found. The idea of not knowing the structure of the tree is that it can't, for example, assume a known bound on the depth, counts of the number of nodes in subtrees, etc., without searching through the tree to find this information directly.

Solution

We'll use rejection sampling. The idea is to choose a node in the infinite binary tree with probability proportional to α^{-d}, and then repeat the process if we picked a node that doesn't actually exist. Conditioned on finding a node i that exists, its probability will be $\frac{\alpha^{\operatorname{depth}(x)}}{\sum_j \alpha^{-\operatorname{depth}(j)}}$.

If we think of a node in the infinite tree as indexed by a binary strength of length equal to its depth, we can generate it by first choosing the length X and then choosing the bits in the string. We want $\Pr[X = n]$ to be proportional to $2^n \alpha^{-n} = (2/\alpha)^n$. Summing the geometric series gives

$$\Pr[X = n] = \frac{(2/\alpha)^n}{1 - (2/\alpha)}.$$

This is a geometric distribution, so we can sample it by repeatedly flipping a biased coin. The number of such coin-flips is $O(X)$, as is the number of random bits we need to generate and the number of tree nodes we need to check. The *expected* value of X is given by the infinite series

$$\sum_{n=0}^{\infty} \frac{(2/\alpha)^n n}{1 - (2/\alpha)};$$

this series converges to some constant by the ratio test.

So each probe costs $O(1)$ time on average, and has at least a constant probability of success, since we choose the root with $\Pr[X = 0] = \frac{1}{1-2/\alpha}$. Using Wald's equation (9.4.1), the total expected time to run the algorithm is $O(1)$.

This is a little surprising, since the output of the algorithm may have more than constant length. But we are interested in expectation, and when $\alpha > 2$ most of the weight lands near the top of the tree.

F.6 Assignment 6: due Friday, 2013-04-26, at 17:00

F.6.1 Increasing subsequences

Let $S_1, \ldots, S_m$ be sets of indices in the range $1 \ldots n$. Say that a permutation π of $1 \ldots n$ is increasing on S_j if $\pi(i_1) < \pi(i_2) < \cdots < \pi(i_{k_j})$ where $i_1 < i_2 < \cdots < i_{k_j}$ are the elements of S_j.

Given a fully polynomial-time randomized approximation scheme that takes as input n and a sequence of sets $S_1, \ldots, S_m$, and approximates the number of permutations π that are increasing on at least one of the S_j.

Solution

This can be solved using a fairly straightforward application of Karp-Luby [KL85] (see §11.4). Recall that for Karp-Luby we need to be able to express our target set U as the union of a polynomial number of covering sets U_j, where we can both compute the size of each U_j and sample uniformly from it. We can then estimate $|U| = \sum_{j,x \in U_j} f(j,x) = \left(\sum_j |U_j|\right) \Pr[f(j,x) = 1]$ where $f(j,x)$ is the indicator for the event that $x \notin U_{j'}$ for any $j' < j$ and in the probability, the pair (j,x) is chosen uniformly at random from $\{(j,x) \mid x \in U_j\}$.

In this case, let U_j be the set of all permutations that are increasing on S_j. We can specify each such permutation by specifying the choice of which $k_j = |S_j|$ elements are in positions $i_1 \dots i_{k_j}$ (the order of these elements is determined by the requirement that the permutation be increasing on S_j) and specifying the order of the remaining $n - k_j$ elements. This gives $\binom{n}{k_j}(n-k_j)! = (n)_{n-k_j}$ such permutations. Begin by computing these counts for all S_j, as well as their sum.

We now wish to sample uniformly from pairs (j,π) where each π is an element of S_j. First sample each j with probability $|S_j| / \sum_\ell |S_\ell|$, using the counts we've already computed. Sampling a permutation uniformly from S_j mirrors the counting argument: choose a k_j-subset for the positions in S_j, then order the remaining elements randomly. The entire sampling step can easily be done in $O(n+m)$ time.

Computing $f(j,\pi)$ requires testing π to see if it is increasing for any $S_{j'}$ for $j < j'$; without doing anything particularly intelligent, this takes $O(nm)$ time. So we can construct and test one sample in $O(nm)$ time. Since each sample has at least a $\rho = 1/m$ chance of having $f(j,\pi) = 1$, from Lemma 11.2.1 we need $O\left(\frac{1}{\epsilon^2 \rho} \log \frac{1}{\delta}\right) = O\left(m\epsilon^{-2} \log \frac{1}{\delta}\right)$ samples to get relative error with probability at least $1-\delta$, for a total cost of $O\left(m^2 n \epsilon^{-2} \log \frac{1}{\delta}\right)$.

F.6.2 Futile word searches

A **word search puzzle** consists of an $n \times n$ grid of letters from some alphabet Σ, where the goal is to find contiguous sequences of letters in one of the eight orthogonal or diagonal directions that form words from some lexicon. For example, in Figure F.2, the left grid contains an instance of `aabc` (running up and left from the rightmost `a` character on the last line), while the right grid contains no instances of this word.

For this problem, you are asked to build an algorithm for constructing

```
bacab  bacab
ccaac  ccaac
bbbac  babac
bbaaa  bbaaa
acbab  acbab
```

Figure F.2: Non-futile (left) and futile (right) word search grids for the lexicon $\{\texttt{aabc}, \texttt{ccca}\}$

word search puzzles with no solution for a given lexicon. That is, given a set of words S over some alphabet and a grid size n, the output should be an $n \times n$ grid of letters such that no word in S appears as a contiguous sequence of letters in one of the eight directions anywhere in the grid. We will refer to such puzzles as **futile word search puzzles**.

1. Suppose the maximum length of any word in S is k. Let p_S be the probability that some word in S is a prefix of an infinite string generated by picking letters uniformly and independently from Σ. Show that there is a constant $c > 0$ such that for any k, Σ, and S, $p_S < ck^{-2}$ implies that there exists, for all n, an $n \times n$ futile word search puzzle for S using only letters from Σ.

2. Give an algorithm that constructs a futile word search puzzle given S and n in expected time polynomial in $|S|$, k, and n, provided $p_S < ck^{-2}$ as above.

Solution

1. We'll apply the symmetric version of the Lovász local lemma. Suppose the grid is filled in independently and uniformly at random with characters from Σ. Given a position ij in the grid, let A_{ij} be the event that there exists a word in S whose first character is at position ij; observe that $\Pr[A_{ij}] \leq 8p_S$ by the union bound (this may be an overestimate, both because we might run off the grid in some directions and because the choice of initial character is not independent). Observe also that A_{ij} is independent of any event $A_{i'j'}$ where $|i - i'| \geq 2k-1$ or $|j - j'| \geq 2k - 1$, because no two words starting at these positions can overlap. So we can build a dependency graph with $p \leq 8p_S$ and $d \leq (4k-3)^2$. The Lovász local lemma shows that there exists an assignment

where no A_{ij} occurs provided $ep(d+1) < 1$ or $8ep_S((4k-3)^2+1) < 1$. This easily holds if $p_S < \frac{1}{8e(4k^2)} = \frac{1}{128e}k^{-2}$.

2. For this part, we can just use Moser-Tardos [MT10], particularly the symmetric version described in Corollary 13.3.4. We have a collection of $m = O(n^2)$ bad events, with $d = \Theta(k^2)$, so the expected number of resamplings is bounded by $m/d = O(n^2/k^2)$. Each resampling requires checking every position in the new grid for an occurrence of some string in S; this takes $O(n^2k \cdot |S|)$ time per resampling even if we are not very clever about it. So the total expected cost is $O(n^4 \cdot |S|/k)$.

   With some more intelligence, this can be improved. We don't need to recheck any position at distance greater than k from any of the at most k letters we resample, and if we are sensible, we can store S using a radix tree or some similar data structure that allows us to look up all words that appear as a prefix of a given length-k string in time $O(k)$. This reduces the cost of each resampling to $O(k^3)$, with an additive cost of $O(k \cdot |S|)$ to initialize the data structure. So the total expected cost is now $O(n^2k + |S|)$.

F.6.3 Balance of power

Suppose you are given a set of n MMORPG players, and a sequence of subsets $S_1, S_2, \ldots, S_m$ of this set, where each subset S_i gives the players who will participate in some raid. Before any of the raids take place, you are to assign each player permanently to one of three factions. If for any i, $|S_i|/2$ or more of the players are in the same faction, then instead of carrying out the raid they will overwhelm and rob the other participants.

Give a randomized algorithm for computing a faction assignment that prevents this tragedy from occurring (for all i) and thus allows all m raids to be completed without incident, assuming that $m > 1$ and $\min_i |S_i| \geq c \ln m$ for some constant $c > 0$ that does not depend on n or m. Your algorithm should run in expected time polynomial in n and m.

Solution

Assign each player randomly to a faction. Let X_{ij} be the number of players in S_i that are assigned to faction j. Then $\mathrm{E}[X_{ij}] = |S_i|/3$. Applying the

Chernoff bound (5.2.2), we have

$$\begin{aligned}\Pr\left[X_{ij} \geq |S_i|/2\right] &= \Pr\left[X_{ij} \geq \left(1+\frac{1}{2}\right)\mathrm{E}\left[X_{ij}\right]\right] \\ &\leq \exp\left(-\left(|S_i|/3\right)\left(\frac{1}{2}\right)^2/3\right) \\ &= e^{-|S_i|/36}.\end{aligned}$$

Let $c = 3 \cdot 36 = 108$. Then if $\min_i |S_i| \geq c \ln m$, for each i, j, it holds that $\Pr\left[X_{ij} \geq |S_i|/2\right] \leq e^{-3\ln m} = m^{-3}$. So the probability that this bound is exceeded for any i and j is at most $(3m)m^{-2} = 3/m^2$. So a random assignment works with at least $1/4$ probability for $m > 1$.

We can generate and test each assignment in $O(nm)$ time. So our expected time is $O(nm)$.

F.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

F.7.1 Dominating sets

A **dominating set** in a graph is a subset S of the vertices for which every vertex v is either in S or adjacent to a vertex in S.

Show that for any graph G, there is an aperiodic, irreducible Markov chain on the dominating sets of G, such that (a) the transition rule for the chain can be implemented in polynomial time; and (b) the stationary distribution of the chain is uniform. (You do not need to say anything about the convergence rate of this chain.)

Solution

Suppose we are have state S_t at time t. We will use a random walk where we choose a vertex uniformly at random to add to or remove from S_t, and carry out the action only if the resulting set is still a dominating set.

In more detail: For each vertex v, with probability $1/n$, $S_{t+1} = S_t \cup \{v\}$ if $v \notin S_t$, $S_{t+1} = S_t \setminus \{v\}$ if $v \in S_t$ and $S_t \setminus \{v\}$ is a dominating set, and $S_{t+1} = S_t$ otherwise. To implement this transition rule, we need to be able

to choose a vertex v uniformly at random (easy) and test in the case where $v \in S_t$ if $S_t \setminus \{v\}$ is a dominating set (also polynomial: for each vertex, check if it or one of its neighbors is in $S_t \setminus \{v\}$, which takes time $O(|V| + |E|)$). Note that we do not need to check if $S_t \cup \{v\}$ is a dominating set.

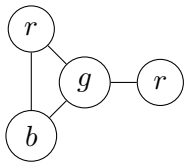
For any pair of adjacent states S and $S' = S \setminus \{v\}$ the probability of moving from S to S' and the probability of moving from S' to S are both $1/n$. So the Markov chain is reversible with a uniform stationary distribution.

This is an aperiodic chain, because there exist minimal dominating sets for which there is a nonzero chance that $S_{t+1} = S_t$.

It is irreducible, because for any dominating set S, there is a path to the complete set of vertices V by adding each vertex in $V \setminus S$ one at a time. Conversely, removing these vertices from V gives a path to S. This gives a path $S \rightsquigarrow V \rightsquigarrow T$ between any two dominating sets S and T.

F.7.2 Tricolor triangles

Suppose you wish to count the number of assignments of colors $\{r, g, b\}$ to nodes of a graph G that have the property that some triangle in G contains all three colors. For example, the four-vertex graph shown below is labeled with one of 18 such colorings (6 permutations of the colors of the triangle nodes times 3 unconstrained choices for the degree-1 node).



Give a fully polynomial-time randomized approximation scheme for this problem.

Solution

Though it is possible, and tempting, to go after this using Karp-Luby (see §11.4), naive sampling is enough.

If a graph has at least one triangle (which can be checked in $O(n^3)$ time just by enumerating all possible triangles), then the probability that that particular triangle is tricolored when colors are chosen uniformly and independently at random is $6/27 = 2/9$. This gives a constant hit rate ρ, so by Lemma 11.2.1, we can get ϵ relative error with $1 - \delta$ probability using

$O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ samples. Each sample costs $O(n^3)$ time to evaluate (again, brute-force checking of all possible triangles), for a total cost of $O\left(n^3\epsilon^{-2}\log\frac{1}{\delta}\right)$.

F.7.3 The n rooks problem

The n rooks problem requires marking as large a subset as possible of the squares in an $n \times n$ grid, so that no two squares in the same row or column are marked.[6]

Consider the following randomized algorithm that attempts to solve this problem:

1. Give each of the n^2 squares a distinct label using a uniformly chosen random permutation of the integers $1 \ldots n^2$.

2. Mark any square whose label is larger than any other label in its row and column.

What is the expected number of marked squares?

Solution

Each square is marked if it is the largest of the $2n-1$ total squares in its row and column. By symmetry, each of these $2n-1$ squares is equally likely to be the largest, so the probability that a particular square is marked is exactly $\frac{1}{2n-1}$. By linearity of expectation, the total expected number of marked squares is then $\frac{n^2}{2n-1}$.

F.7.4 Pursuing an invisible target on a ring

Suppose that you start at position 0 on a ring of size $2n$, while a target particle starts at position n. At each step, starting at position i, you can choose whether to move to any of positions $i-1$, i, or $i+1$. At the same time, the target moves from its position j to either $j-1$ or $j+1$ with equal probability, independent of its previous moves or your moves. Aside from knowing that the target starts at n at time 0, you cannot tell where the target moves.

Your goal is to end up on the same node as the target after some step.[7] Give an algorithm for choosing your moves such that, for any $c > 0$, you

[6] This is not actually a hard problem.

[7] Note that you must be in the same place at the end of the step: if you move from 1 to 2 while the target moves from 2 to 1, that doesn't count.

encounter the target in at most $2n$ steps with probability at least $1 - n^{-c}$ for sufficiently large n.

Solution

The basic idea is to just go through positions $0, 1, 2, \ldots$ until we encounter the target, but we have to be a little careful about parity to make sure we don't pass it by accident.[8]

Let $X_i = \pm 1$ be the increment of the target's i-th move, and let $S_i = \sum_{j=1}^{i} X_j$, so that its position after i steps is $n + S_i \bmod 2n$.

Let Y_i be the position of the pursuer after i steps.

First move: stay at 0 if n is odd, move to 1 if n is even. The purpose of this is to establish the invariant that $n + S_i - Y_i$ is even starting at $i = 1$. For subsequent moves, let $Y_{i+1} = Y_i + 1$. Observe that this maintains the invariant.

We assume that n is at least 2. This is necessary to ensure that at time 1, $Y_1 \leq n + S_1$.

Claim: if at time $2n$, $Y_{2n} \geq n + S_{2n}$, then at some time $i \leq 2n$, $Y_i = n + S_i$. Proof: Let i be the first time at which $Y_i \geq n + S_i$; under the assumption that $n \geq 2$, $i \geq 1$. So from the invariant, we can't have $Y_i = n + S_i + 1$, and if $Y_i \geq n + S_i + 2$, we have $Y_{i-1} \geq Y_i - 1 \geq n + S_i + 1 \geq n + S_{i-1}$, contradicting our assumption that i is minimal. The remaining alternative is that $Y_i = n + S_i$, giving a collision at time i.

We now argue that $Y_{2n} \geq n - 1$ is very likely to be at least $n + S_{2n}$. Since S_{2n} is a sum of $2n$ independent ± 1 variables, from Hoeffding's inequality we have $\Pr[Y_n < n + S_{2n}] \leq \Pr[S_{2n} \geq n] \leq e^{-n^2/4n} = e^{-n/4}$. For sufficiently large n, this is much smaller than n^{-c} for any fixed c.

[8] This is not the only possible algorithm, but there are a lot of plausible-looking algorithms that turn out not to work. One particularly tempting approach is to run to position n using the first n steps and then spend the next n steps trying to hit the target in the immediate neighborhood of n, either by staying put (a sensible strategy when lost in the woods in real life, assuming somebody is looking for you), or moving in a random walk of some sort starting at n. This doesn't work if we want a high-probability bound. To see this, observe that the target has a small but nonzero constant probability in the limit of begin at some position greater than or equal to $n + 4\sqrt{n}$ after exactly $n/2$ steps. Conditioned on starting at $n + 4\sqrt{n}$ or above, its chances of moving below $n + 4\sqrt{n} - 2\sqrt{n} = n + 2\sqrt{n}$ at any time in the next $3n/2$ steps is bounded by $e^{-4n/2(3n/2)} = e^{-4/3}$ (Azuma), and a similar bound holds independently for our chances of getting up to $n + 2\sqrt{n}$ or above. Multiplying out all these constants gives a constant probability of failure. A similar but bigger disaster occurs if we don't rush to n first.

Appendix G

Sample assignments from Spring 2011

G.1 Assignment 1: due Wednesday, 2011-01-26, at 17:00

G.1.1 Bureaucratic part

Send me email! My address is james.aspnes@gmail.com.

In your message, include:

1. Your name.
2. Your status: whether you are an undergraduate, grad student, auditor, etc.
3. Anything else you'd like to say.

(You will not be graded on the bureaucratic part, but you should do it anyway.)

G.1.2 Rolling a die

The usual model of a randomized algorithm assumes a source of fair, independent random bits. This makes it easy to generate uniform numbers in the range $0 \ldots 2^n - 1$, but not so easy for other ranges. Here are two algorithms for generating a uniform random integer $0 \leq s < n$:

- **Rejection sampling** generates a uniform random integer $0 \le s < 2^{\lceil \lg n \rceil}$. If $s < n$, return s; otherwise keep trying until you get some $s < n$.
- **Arithmetic coding** or **range coding** generates a sequence of bits $r_1, r_2, \ldots r_k$ until the half-open interval $[\sum_{i=1}^{k} 2^{-i} r_i, \sum_{i=1}^{k} 2^{-i} r_i + 2^{-k-1})$ is a subset of $[s/n, (s+1)/n)$ for some s; it then returns s.

1. Show that both rejection sampling and range coding produce a uniform value $0 \le s < n$ using an expected $O(\log n)$ random bits.
2. Which algorithm has a better constant?
3. Does there exist a function f and an algorithm that produces a uniform value $0 \le s < n$ for any n using $f(n)$ random bits with probability 1?

Solution

1. For rejection sampling, each sample requires $\lceil \lg n \rceil$ bits and is accepted with probability $n/2^{\lceil \lg n \rceil} \ge 1/2$. So rejection sampling returns a value after at most 2 samples on average, using no more than an expected $2\lceil \lg n \rceil < 2(\lg n + 1)$ expected bits for the worst n.

   For range coding, we keep going as long as one of the $n-1$ nonzero endpoints s/n lies inside the current interval. After k bits, the probability that one of the 2^k intervals contains an endpoint is at most $(n-1)2^{-k}$; in particular, it drops below 1 as soon as $k = 2^{\lceil \lg n \rceil}$ and continues to drop by $1/2$ for each additional bit, requiring 2 more bits on average. So the expected cost of range coding is at most $\lceil \lg n \rceil + 2 < \lg n + 3$ bits.

2. We've just shown that range coding beats rejection sampling by a factor of 2 in the limit, for worst-case n. It's worth noting that other factors might be more important if random bits are cheap: rejection sampling is much easier to code and avoids the need for division.

3. There is no algorithm that produces a uniform value $0 \le s < n$ for all n using any fixed number of bits. Suppose such an algorithm existed. Fix some n. For all n values s to be equally likely, the sets of random bits $M^{-1}(s) = \{r \mid M(r) = s\}$ must have the same size. But this can only happen if n divides $2^{f(n)}$, which works only for n a power of 2.

G.1.3 Rolling many dice

Suppose you repeatedly roll an n-sided die. Give an asymptotic (big-Θ) bound on the expected number of rolls until you roll some number you have already rolled before.

Solution

In principle, it is possible to compute this value exactly, but we are lazy.

For a lower bound, observe that after m rolls, each of the $\binom{m}{2}$ pairs of rolls has probability $1/n$ of being equal, for an expected total of $\binom{m}{2}/n$ duplicates. For $m = \sqrt{n}/2$, this is less than $1/8$, which shows that the expected number of rolls is $\Omega(\sqrt{n})$.

For the upper bound, suppose we have already rolled the die $\sqrt{n}$ times. If we haven't gotten a duplicate already, each new roll has probability at least $\sqrt{n}/n = 1/\sqrt{n}$ of matching a previous roll. So after an additional $\sqrt{n}$ rolls on average, we get a repeat. This shows that the expected number of rolls is $O(\sqrt{n})$.

Combining these bounds shows that we need $\Theta(\sqrt{n})$ rolls on average.

G.1.4 All must have candy

A set of n_0 children each reach for one of n_0 candies, with each child choosing a candy independently and uniformly at random. If a candy is chosen by exactly one child, the candy and child drop out. The remaining n_1 children and candies then repeat the process for another round, leaving n_2 remaining children and candies, etc. The process continues until ever child has a candy.

Give the best bound you can on the expected number of rounds until every child has a candy.

Solution

Let $T(n)$ be the expected number of rounds remaining given we are starting with n candies. We can set up a probabilistic recurrence relation $T(n) = 1 + T(n - X_n)$ where X_n is the number of candies chosen by eactly one child. It is easy to compute $\mathrm{E}[X_n]$, since the probability that any candy gets chosen exactly once is $n(1/n)(1 - 1/n)^{n-1} = (1 - 1/n)^{n-1}$. Summing over all candies gives $\mathrm{E}[X_n] = n(1 - 1/n)^{n-1}$.

The term $(1-1/n)^{n-1}$ approaches e^{-1} in the limit, so for any fixed $\epsilon > 0$, we have $n(1 - 1/n)^{n-1} \geq n(e^{-1} - \epsilon)$ for sufficiently large n. We can get a quick bound by choosing ϵ so that $e^{-1} - \epsilon \geq 1/4$ (for example) and then

applying the Karp-Upfal-Wigderson inequality (I.3.1) with $\mu(n) = n/4$ to get

$$\begin{aligned}\mathrm{E}\left[T(n)\right] &\leq \int_1^n \frac{1}{t/4}\,dt \\ &= 4\ln n.\end{aligned}$$

There is a sneaky trick here, which is that we stop if we get down to 1 candy instead of 0. This avoids the usual problem with KUW and $\ln 0$, by observing that we can't ever get down to exactly one candy: if there were exactly one candy that gets grabbed twice or not at all, then there must be some other candy that also gets grabbed twice or not at all.

This analysis is sufficient for an asymptotic estimate: the last candy gets grabbed in $O(\log n)$ rounds on average. For most computer-sciency purposes, we'd be done here.

We can improve the constant slightly by observing that $(1 - 1/n)^{n-1}$ is in fact always greater than or equal to e^{-1}. The easiest way to see this is to plot the function, but if we want to prove it formally we can show that $(1-1/n)^{n-1}$ is a decreasing function by taking the derivative of its logarithm:

$$\begin{aligned}\frac{d}{dn}\ln(1-1/n)^{n-1} &= \frac{d}{dn}(n-1)\ln(1-1/n) \\ &= \ln(1-1/n) + \frac{n-1}{1-1/n}\cdot\frac{-1}{n^2}.\end{aligned}$$

and observing that it is negative for $n > 1$ (we could also take the derivative of the original function, but it's less obvious that it's negative). So if it approaches e^{-1} in the limit, it must do so from above, implying $(1-1/n)^{n-1} \geq e^{-1}$.

This lets us apply (I.3.1) with $\mu(n) = n/e$, giving $\mathrm{E}\left[T(n)\right] \leq e\ln n$.

If we skip the KUW bound and use the analysis in §I.4.2 instead, we get that $\Pr\left[T(n) \geq \ln n + \ln(1/\epsilon)\right] \leq \epsilon$. This suggests that the actual expected value should be $(1 + o(1))\ln n$.

G.2 Assignment 2: due Wednesday, 2011-02-09, at 17:00

G.2.1 Randomized dominating set

A **dominating set** in a graph $G = (V, E)$ is a set of vertices D such that each of the n vertices in V is either in D or adjacent to a vertex in D.

Suppose we have a d-regular graph, in which every vertex has exactly d neighbors. Let D_1 be a random subset of V in which each vertex appears with independent probability p. Let D be the union of D_1 and the set of all vertices that are not adjacent to any vertex in D_1. (This construction is related to a classic maximal independent set algorithm of Luby [Lub85], and has the desirable property in a distributed system of finding a dominating set in only one round of communication.)

1. What would be a good value of p if our goal is to minimize $\mathrm{E}[|D|]$, and what bound on $\mathrm{E}[|D|]$ does this value give?

2. For your choice of p above, what bound can you get on $\Pr[|D| - \mathrm{E}[|D|] \geq t]$?

Solution

1. First let's compute $\mathrm{E}[|D|]$. Let X_v be the indicator for the event that $v \in D$. Then $X_v = 1$ if either (a) v is in D_1, which occurs with probability p; or (b) v and all d of its neighbors are not in D_1, which occurs with probability $(1-p)^{d+1}$. Adding these two cases gives $\mathrm{E}[X_v] = p + (1-p)^{d+1}$ and thus

$$\mathrm{E}[|D|] = \sum_v \mathrm{E}[X_v] = n\left(p + (1-p)^{d+1}\right). \tag{G.2.1}$$

   We optimize $\mathrm{E}[|D|]$ in the usual way, by seeking a minimum for $\mathrm{E}[X_v]$. Differentiating with respect to p and setting to 0 gives $1 - (d+1)(1-p)^d = 0$, which we can solve to get $p = 1 - (d+1)^{-1/d}$. (We can easily observe that this must be a minimum because setting p to either 0 or 1 gives $\mathrm{E}[X_v] = 1$.)

   The value of $\mathrm{E}[|D|]$ for this value of p is the rather nasty expression $n(1 - (d+1)^{-1/d} + (d+1)^{-1-1/d})$.

   Plotting the d factor up suggests that it goes to $\ln d/d$ in the limit, and both Maxima and www.wolframalpha.com agree with this. Knowing

the answer, we can prove it by showing

$$\begin{aligned}
\lim_{d\to\infty} \frac{1-(d+1)^{-1/d}+(d+1)^{-1-1/d}}{\ln d/d} &= \lim_{d\to\infty} \frac{1-d^{-1/d}+d^{-1-1/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-d^{-1/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-e^{-\ln d/d}}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{1-\left(1-\ln d/d+O(\ln^2 d/d^2)\right)}{\ln d/d} \\
&= \lim_{d\to\infty} \frac{\ln d/d+O(\ln^2 d/d^2)}{\ln d/d} \\
&= \lim_{d\to\infty} (1+O(\ln d/d)) \\
&= 1.
\end{aligned}$$

This lets us write $\mathrm{E}\left[|D|\right] = (1+o(1))n \ln d/d$, where we bear in mind that the $o(1)$ term depends on d but not n.

2. Suppose we fix X_v for all but one vertex u. Changing X_u from 0 to 1 can increase $|D|$ by at most one (if u wasn't already in D) and can decrease it by at most $d-1$ (if u wasn't already in D and adding u to D_1 lets all d of u's neighbors drop out). So we can apply the method of bounded differences with $c_i = d-1$ to get

$$\Pr\left[|D| - \mathrm{E}\left[|D|\right] \geq t\right] \leq \exp\left(-\frac{t^2}{2n(d-1)^2}\right).$$

A curious feature of this bound is that it doesn't depend on p at all. It may be possible to get a tighter bound using a better analysis, which might pay off for very large d (say, $d \gg \sqrt{n}$).

G.2.2 Chernoff bounds with variable probabilities

Let $X_1 \ldots X_n$ be a sequence of Bernoulli random variables, where for all i, $\mathrm{E}\left[X_i \mid X_1 \ldots X_{i-1}\right] \leq p_i$. Let $S = \sum_{i=1}^n X_i$ and $\mu = \sum_{i=1}^n p_i$. Show that, for all $\delta \geq 0$,

$$\Pr\left[S \geq (1+\delta)\mu\right] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu.$$

Solution

Let $S_t = \sum_{i=1}^{t} X_i$, so that $S = S_n$, and let $\mu_t = \sum_{i=1}^{t} p_i$. We'll show by induction on t that $\mathrm{E}\left[e^{\alpha S_t}\right] \leq \exp(e^{\alpha-1}\mu_t)$, when $\alpha > 0$.

Compute

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha S}\right] &= \mathrm{E}\left[e^{\alpha S_{n-1}} e^{\alpha X_n}\right] \\
&= \mathrm{E}\left[e^{\alpha S_{n-1}} \mathrm{E}\left[e^{\alpha X_n} \;\middle|\; X_1, \ldots, X_{n-1}\right]\right] \\
&= \mathrm{E}\left[e^{\alpha S_{n-1}} \left(\Pr\left[X_n = 0 \mid X_1, \ldots, X_{n-1}\right] + e^{\alpha} \Pr\left[X_n = 1 \mid X_1, \ldots, X_{n-1}\right]\right)\right] \\
&= \mathrm{E}\left[e^{\alpha S_{n-1}} \left(1 + (e^{\alpha} - 1) \Pr\left[X_n = 1 \mid X_1, \ldots, X_{n-1}\right]\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}} \left(1 + (e^{\alpha} - 1) p_n\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}} \exp\left((e^{\alpha} - 1) p_n\right)\right] \\
&\leq \mathrm{E}\left[e^{\alpha S_{n-1}}\right] \exp\left((e^{\alpha} - 1) p_n\right) \\
&\leq \exp\left(e^{\alpha} - 1\right)\mu_{n-1} \exp\left((e^{\alpha} - 1) p_n\right) \\
&= \exp\left(e^{\alpha} - 1\right)\mu_n .
\end{aligned}$$

Now apply the rest of the proof of (5.2.1) to get the full result.

G.2.3 Long runs

Let W be a binary string of length n, generated uniformly at random. Define a **run** of ones as a maximal sequence of contiguous ones; for example, the string 11100110011111101011 contains 5 runs of ones, of length 3, 2, 6, 1, and 2.

Let X_k be the number of runs in W of length k or more.

1. Compute the exact value of $\mathrm{E}[X_k]$ as a function of n and k.
2. Give the best concentration bound you can for $|X_k - \mathrm{E}[X_k]|$.

Solution

1. We'll compute the probability that any particular position $i = 1 \ldots n$ is the start of a run of length k or more, then sum over all i. For a run of length k to start at position i, either (a) $i = 1$ and $W_i \ldots W_{i+k-1}$ are all 1, or (b) $i > 1$, $W_{i-1} = 0$, and $W_i \ldots W_{i+k-1}$ are all 1. Assuming $n \geq k$, case (a) adds 2^{-k} to $\mathrm{E}[X_k]$ and case (b) adds $(n-k)2^{-k-1}$, for a total of $2^{-k} + (n-k)2^{-k-1} = (n-k+2)2^{-k-1}$.

2. We can get an easy bound without too much cleverness using McDiarmid's inequality (5.3.13). Observe that X_k is a function of the independent random variables $W_1 \dots W_n$ and that changing one of these bits changes X_k by at most 1 (this can happen in several ways: a previous run of length $k-1$ can become a run of length k or vice versa, or two runs of length k or more separated by a single zero may become a single run, or vice versa). So (5.3.13) gives $\Pr\left[|X - \mathrm{E}\left[X\right]| \geq t\right] \leq 2\exp\left(-\frac{t^2}{2n}\right)$.

   We can improve on this a bit by grouping the W_i together into blocks of length ℓ. If we are given control over a block of ℓ consecutive bits and want to minimize the number of runs, we can either (a) make all the bits zero, causing no runs to appear within the block and preventing adjacent runs from extending to length k using bits from the block, or (b) make all the bits one, possibly creating a new run but possibly also causing two existing runs on either side of the block to merge into one. In the first case, changing all the bits to one except for a zero after every k consecutive ones creates at most $\lfloor\frac{\ell+2k-1}{k+1}\rfloor$ new runs. Treating each of the $\lceil n\rceil\ell$ blocks as a single variable then gives $\Pr\left[|X - \mathrm{E}\left[X\right]| \geq t\right] \leq 2\exp\left(-\frac{t^2}{2\lceil n/\ell\rceil(\lfloor(\ell+2k-1)/(k+1)\rfloor)^2}\right)$. Staring at plots of the denominator for a while suggests that it is minimized at $\ell = k+3$, the largest value with $\lfloor(\ell+2k-1)/(k+1)\rfloor \leq 2$. This gives $\Pr\left[|X - \mathrm{E}\left[X\right]| \geq t\right] \leq 2\exp\left(-\frac{t^2}{8\lceil n/(k+3)\rceil}\right)$, improving the bound on t from $\Theta(\sqrt{n\log(1/\epsilon)})$ to $\Theta(\sqrt{(n/k)\log(1/\epsilon)})$.

   For large k, the expectation of any individual X_k becomes small, so we'd expect that Chernoff bounds would work better on the upper bound side than the method of bounded differences. Unfortunately, we don't have independence. But from Problem G.2.2, we know that the usual Chernoff bound works as long as we can show $\mathrm{E}\left[X_i \mid X_1, \dots, X_{i-1}\right] \leq p_i$ for some sequence of fixed bounds p_i.

   For X_1, there are no previous X_i, and we have $\mathrm{E}\left[X_1\right] = 2^{-k}$ exactly.

   For X_i with $i > 1$, fix $X_1, \dots, X_{i-1}$; that is, condition on the event $X_j = x_j$ for all $j < i$ with some fixed sequence $x_1, \dots, x_{i-1}$. Let's call this event A. Depending on the particular values of the x_j, it's not clear how conditioning on A will affect X_i; but we can split on the value of W_{i-1} to show that either it has no effect or $X_i = 0$:

$$\begin{aligned}\mathrm{E}\left[X_i \mid A\right] &= \mathrm{E}\left[X_i \mid A, W_{i-1} = 0\right]\Pr\left[W_{i-1} = 0 \mid A\right] + \mathrm{E}\left[X_i \mid A, W_{i-1} = 1\right]\Pr\left[W_{i-1} = 1 \mid A\right] \\ &\leq 2^{-k}\Pr\left[W_{i-1} = 0 \mid A\right] + 0 \\ &\leq 2^{-k}.\end{aligned}$$

So we have $p_i \leq 2^{-k}$ for all $1 \leq i \leq n-k+1$. This gives $\mu = 2^{-k}(n-k+1)$, and $\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$.

If we want a two-sided bound, we can set $\delta = 1$ (since X can't drop below 0 anyway, and get $\Pr\left[|X - \mathrm{E}[X]| > 2^{-k}(n-k+1)\right] \leq \left(\frac{e}{4}\right)^{2^{-k}(n-k+1)}$. This is exponentially small for $k = o(\lg n)$. If k is much bigger than $\lg n$, then we have $\mathrm{E}[X] \ll 1$, so Markov's inequality alone gives us a strong concentration bound.

However, in both cases, the bounds are competitive with the previous bounds from McDiarmid's inequality only if $\mathrm{E}[X] = O(\sqrt{n \log(1/\epsilon)})$. So McDiarmid's inequality wins for $k = o(\log n)$, Markov's inequality wins for $k = \omega(\log n)$, and Chernoff bounds may be useful for a small interval in the middle.

G.3 Assignment 3: due Wednesday, 2011-02-23, at 17:00

G.3.1 Longest common subsequence

A **common subsequence** of two sequences v and w is a sequence u of length k such that there exist indices $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$ with $u_\ell = v_{i_\ell} = w_{j_\ell}$ for all ℓ. For example, `ardab` is a common subsequence of `abracadabra` and `cardtable`.

Let v and w be words of length n over an alphabet of size n drawn independently and uniformly at random. Give the best upper bound you can on the expected length of the longest common subsequence of v and w.

Solution

Let's count the expectation of the number X_k of common subsequences of length k. We have $\binom{n}{k}$ choices of positions in v, and $\binom{n}{k}$ choices of positions in w; for each such choices, there is a probability of exactly n^{-k} that the

corresponding positions match. This gives

$$\begin{aligned}\mathrm{E}[X_k] &= \binom{n}{k}^2 n^{-k} \\ &< \frac{n^k}{(k!)^2}. \\ &< \frac{n^k}{(k/e)^{2k}} \\ &= \left(\frac{ne^2}{k^2}\right)^k.\end{aligned}$$

We'd like this bound to be substantially less than 1. We can't reasonably expect this to happen unless the base of the exponent is less than 1, so we need $k > e\sqrt{n}$.

If $k = (1+\epsilon)e\sqrt{n}$ for any $\epsilon > 0$, then $\mathrm{E}[X_k] < (1+\epsilon)^{-2e\sqrt{n}} < \frac{1}{n}$ for sufficiently large n. It follows that the expected length of the longest common subsequent is at most $(1+\epsilon)e\sqrt{n}$ for sufficiently large n (because if there are no length-k subsequences, the longest subsequence has length at most $k-1$, and if there is at least one, the longest has length at most n; this gives a bound of at most $(1-1/n)(k-1) + (1/n)n < k$). So in general we have the length of the longest common subsequence is at most $(1+o(1))e\sqrt{n}$.

Though it is not required by the problem, here is a quick argument that the expected length of the longest common subsequence is $\Omega(\sqrt{n})$, based on the **Erdős-Szekeres theorem** [ES35].[1] The Erdős-Szekeres theorem says that any permutation of n^2+1 elements contains either an increasing sequence of $n+1$ elements or a decreasing sequence of $n+1$ elements. Given two random sequences of length n, let S be the set of all elements that appear in both, and consider two permutations ρ and σ of S corresponding to the order in which the elements appear in v and w, respectively (if an element appears multiple times, pick one of the occurrences at random). Then the Erdős-Szekeres theorem says that ρ contains a sequence of length at least $\lfloor\sqrt{|\rho|-1}\rfloor$ that is either increasing or decreasing with respect to the order given by σ; by symmetry, the probability that it is increasing is at least $1/2$. This gives an expected value for the longest common subsequence that is at least $\mathrm{E}\left[\sqrt{|\rho|-1}\right]/2$.

Let $X = |\rho|$. We can compute a lower bound $\mathrm{E}[X]$ easily; each possible element fails to occur in v with probability $(1-1/n)^n \le e^{-1}$, and similarly for w.

[1] As suggested by Benjamin Kunsberg.

So the chance that an element appears in both sequences is at least $(1-e^{-1})^2$, and thus $\mathrm{E}[X] \geq n(1-e^{-1})^2$. What we want is $\mathrm{E}\left[\sqrt{X-1}\right]/2$; but here the fact that $\sqrt{x}$ is concave means that $\mathrm{E}\left[\sqrt{X-1}\right] \geq \sqrt{\mathrm{E}[X-1]}$ by Jensen's inequality (4.3.1). So we have $\mathrm{E}\left[\sqrt{|\rho|-1}\right]/2 \geq \sqrt{n(1-e^{-1})^2-1}/2 \approx \frac{1-e^{-1}}{2}\sqrt{n}$.

This is not a very good bound (empirically, the real bound seems to be in the neighborhood of $1.9\sqrt{n}$ when $n = 10000$), but it shows that the upper bound of $(1+o(1))e\sqrt{n}$ is tight up to constant factors.

G.3.2 A strange error-correcting code

Let Σ be an alphabet of size $m+1$ that includes m non-blank symbols and a special blank symbol. Let S be a set of $\binom{n}{k}$ strings of length n with non-blank symbols in exactly k positions each, such that no two strings in S have non-blank symbols in the same k positions.

For what value of m can you show S exists such that no two strings in S have the same non-blank symbols in $k-1$ positions?

Solution

This is a job for the Lovász Local Lemma. And it's even symmetric, so we can use the symmetric version (Corollary 13.3.2).

Suppose we assign the non-blank symbols to each string uniformly and independently at random. For each $A \subseteq S$ with $|A| = k$, let X_A be the string that has non-blank symbols in all positions in A. For each pair of subsets A, B with $|A| = |B| = k$ and $|A \cap B| = k-1$, let $C_{A,B}$ be the event that X_A and X_B are identical on all positions in $A \cap B$. Then $\Pr[C_{A,B}] = m^{-k+1}$.

We now now need to figure out how many events are in each neighborhood $\Gamma(C_{A,B})$. Since $C_{A,B}$ depends only on the choices of values for A and B, it is independent of any events $C_{A',B'}$ where neither of A' or B' is equal to A or B. So we can make $\Gamma(C_{A,B})$ consist of all events $C_{A,B'}$ and $C_{A',B}$ where $B' \neq B$ and $A' \neq A$.

For each fixed A, there are exactly $(n-k)k$ events B that overlap it in $k-1$ places, because we can specify B by choosing the elements in $B \setminus A$ and $A \setminus B$. This gives $(n-k)k-1$ events $C_{A,B'}$ where $B' \neq B$. Applying the same argument for A' gives a total of $d = 2(n-k)k-2$ events in $\Gamma(C_{A,B})$. Corollary 13.3.2 applies if $ep(d+1) \leq 1$, which in this case means

$em^{-(k-1)}(2(n-k)k-1) \leq 1$. Solving for m gives

$$m \geq (2e(n-k)k-1)^{1/(k-1)}. \tag{G.3.1}$$

For $k \ll n$, the $(n-k)^{1/(k-1)} \approx n^{1/(k-1)}$ term dominates the shape of the right-hand side asymptotically as k gets large, since everything else goes to 1. This suggests we need $k = \Omega(\log n)$ to get m down to a constant.

Note that (G.3.1) doesn't work very well when $k = 1$.[2] For the $k = 1$ case, there is no overlap in non-blank positions between different strings, so $m = 1$ is enough.

G.3.3 A multiway cut

Given a graph $G = (V, E)$, a **3-way cut** is a set of edges whose endpoints lie in different parts of a partition of the vertices V into three disjoint parts $S \cup T \cup U = V$.

1. Show that any graph with m edges has a 3-way cut with at least $2m/3$ edges.

2. Give an efficient deterministic algorithm for finding such a cut.

Solution

1. Assign each vertex independently to S, T, or U with probability $1/3$ each. Then the probability that any edge uv is contained in the cut is exactly $2/3$. Summing over all edges gives an expected $2m/3$ edges.

2. We'll derandomize the random vertex assignment using the method of conditional probabilities. Given a partial assignment of the vertices, we can compute the conditional expectation of the size of the cut assuming all other vertices are assigned randomly: each edge with matching assigned endpoints contributes 0 to the total, each edge with non-matching assigned endpoints contributes 1, and each edge with zero or one assigned endpoints contributes $2/3$. We'll pick values for the vertices in some arbitrary order to maximize this conditional expectation (since our goal is to get a large cut). At each step, we need only consider the effect on edges incident to the vertex we are assigning whose other endpoints are already assigned, because the contribution of any other edge is not changed by the assignment. Then maximizing

[2] Thanks to Brad Hayes for pointing this out.

the conditional probability is done by choosing an assignment that matches the assignment of the fewest previously-assigned neighbors: in other words, the natural greedy algorithm works. The cost of this algorithm is $O(n+m)$, since we loop over all vertices and have to check each edge at most once for each of its endpoints.

G.4 Assignment 4: due Wednesday, 2011-03-23, at 17:00

G.4.1 Sometimes successful betting strategies are possible

You enter a casino with $X_0 = a$ dollars, and leave if you reach 0 dollars or b or more dollars, where $a, b \in \mathbb{N}$. The casino is unusual in that it offers arbitrary fair games subject to the requirements that:

- Any payoff resulting from a bet must be a nonzero integer in the range $-X_t$ to X_t, inclusive, where X_t is your current wealth.
- The expected payoff must be exactly 0. (In other words, your assets X_t should form a martingale sequence.)

For example, if you have 2 dollars, you may make a bet that pays off -2 with probability 2/5, $+1$ with probability 2/5 and $+2$ with probability 1/5; but you may not make a bet that pays off -3, $+3/2$, or $+4$ under any circumstances, or a bet that pays off -1 with probability 2/3 and $+1$ with probability 1/3.

1. What strategy should you use to maximize your chances of leaving with at least b dollars?
2. What strategy should you use to maximize your changes of leaving with nothing?
3. What strategy should you use to maximize the number of bets you make before leaving?

Solution

1. Let X_t be your wealth at time t, and let τ be the stopping time when you leave. Because $\{X_t\}$ is a martingale, $\mathrm{E}[X_0] = a = \mathrm{E}[X_\tau] = \Pr[X_\tau \geq b] \mathrm{E}[X_\tau \mid X_\tau \geq b]$. So $\Pr[X_\tau \geq b]$ is maximized by making $\mathrm{E}[X_\tau \mid X_\tau \geq b]$ as small as possible. It can't be any smaller than b,

which can be obtained exactly by making only ± 1 bets. The gives a probability of leaving with b of exactly a/b.

2. Here our goal is to minimize $\Pr[X_\tau \geq b]$, so we want to make $\mathrm{E}[X_\tau \mid X_\tau \geq b]$ as large as possible. The largest value of X_τ we can possibly reach is $2(b-1)$; we can obtain this value by betting ± 1 until we reach $b-1$, then making any fair bet with positive payoff $b-1$ (for example, $\pm(b-1)$ with equal probability works, as does a bet that pays off $b-1$ with probability $1/b$ and -1 with probability $(b-1)/b$). In this case we get a probability of leaving with 0 of $1 - \frac{a}{2(b-1)}$.

3. For each t, let $\delta_t = X_t - X_{t-1}$ and $V_t = \mathrm{Var}[\delta_t \mid \mathcal{F}_t]$. We have previously shown (see the footnote to §9.4.1) that $\mathrm{E}[X_\tau^2] = \mathrm{E}[X_0^2] + \mathrm{E}[\sum_{t=1}^{\tau} V_t]$ where τ is an appropriate stopping time. When we stop, we know that $X_\tau^2 \leq (2(b-1))^2$, which puts an upper bound on $\mathrm{E}[\sum_{i=1}^{\tau} V_i]$. We can spend this bound most parsimoniously by minimizing V_i as much as possible. If we make each $\delta_t = \pm 1$, we get the smallest possible value for V_t (since any change contributes at least 1 to the variance). However, in this case we don't get all the way out to $2(b-1)$ at the high end; instead, we stop at b, giving an expected number of steps equal to $a(b-a)$.

   We can do a bit better than this by changing our strategy at $b-1$. Instead of betting ± 1, let's pick some x and place a bet that pays off $b-1$ with probability $\frac{1}{b}$ and -1 with probability $\frac{b-1}{b} = 1 - \frac{1}{b}$. (The idea here is to minimize the conditional variance while still allowing ourselves to reach $2(b-1)$.) Each ordinary random walk step has $V_t = 1$; a "big" bet starting at $b-1$ has $V_t = 1 - \frac{1}{b} + \frac{(b-1)^2}{b} = \frac{b-1+b^2-2b+1}{b} = b - \frac{1}{b}$.

   To analyze this process, observe that starting from a, we first spending $a(b-a-1)$ steps on average to reach either 0 (with probability $1 - \frac{a}{b-1}$ or $b-1$ (with probability $\frac{a}{b-1}$. In the first case, we are done. Otherwise, we take one more step, then with probability $\frac{1}{b}$ we lose and with probability $\frac{b-1}{b}$ we continue starting from $b-2$. We can write a recurrence for our expected number of steps $T(a)$ starting from a, as:

$$T(a) = a(b-a-1) + \frac{a}{b-1}\left(1 + \frac{b-1}{b}T(b-2)\right). \qquad \text{(G.4.1)}$$

When $a = b - 2$, we get

$$\begin{aligned} T(b-2) &= (b-2) + \frac{b-2}{b-1}\left(1 + \frac{b-1}{b}T(b-2)\right) \\ &= (b-2)\left(1 + \frac{1}{b-1}\right) + \frac{b-2}{b}T(b-2), \end{aligned}$$

which gives

$$\begin{aligned} T(b-2) &= \frac{(b-2)\frac{2b-1}{b-1}}{2/b} \\ &= \frac{b(b-2)(2b-1)}{2(b-1)}. \end{aligned}$$

Plugging this back into (G.4.1) gives

$$\begin{aligned} T(a) &= a(b-a-1) + \frac{a}{b-1}\left(1 + \frac{b-1}{b}\frac{b(b-2)(2b-1)}{2(b-1)}\right) \\ &= ab - a^2 + a + \frac{a}{b-1} + \frac{a(b-2)(2b-1)}{2(b-1)} \\ &= \frac{3}{2}ab + O(b). \end{aligned} \tag{G.4.2}$$

This is much better than the $a(b-a)$ value for the straight ± 1 strategy, especially when a is also large.

I don't know if this particular strategy is in fact optimal, but that's what I'd be tempted to bet.

G.4.2 Random walk with reset

Consider a random walk on $\mathbb{N}$ that goes up with probability $1/2$, down with probability $3/8$, and resets to 0 with probability $1/8$. When $X_t > 0$, this gives:

$$X_{t+1} = \begin{cases} X_t + 1 & \text{with probability } 1/2, \\ X_t - 1 & \text{with probability } 3/8, \text{ and} \\ 0 & \text{with probability } 1/8. \end{cases}$$

When $X_t = 0$, we let $X_{t+1} = 1$ with probability $1/2$ and 0 with probability $1/2$.

1. What is the stationary distribution of this process?

2. What is the mean recurrence time μ_n for some state n?

3. Use μ_n to get a tight asymptotic (i.e., big-Θ) bound on $\mu_{0,n}$, the expected time to reach n starting from 0.

Solution

1. For $n > 0$, we have $\pi_n = \frac{1}{2}\pi_{n-1} + \frac{3}{8}\pi_{n+1}$, with a base case $\pi_0 = \frac{1}{8} + \frac{3}{8}\pi_0 + \frac{3}{8}\pi_1$.

   The π_n expression is a linear homogeneous recurrence, so its solution consists of linear combinations of terms b^n, where b satisfies $1 = \frac{1}{2}b^{-1} + \frac{3}{8}b$. The solutions to this equation are $b = 2/3$ and $b = 2$; we can exclude the $b = 2$ case because it would make our probabilities blow up for large n. So we can reasonably guess $\pi_n = a(2/3)^n$ when $n > 0$.

   For $n = 0$, substitute $\pi_0 = \frac{1}{8} + \frac{3}{8}\pi_0 + \frac{3}{8}a(2/3)$ to get $\pi_0 = \frac{1}{5} + \frac{2}{5}a$. Now substitute

$$\begin{aligned}\pi_1 &= (2/3)a \\ &= \frac{1}{2}\pi_0 + \frac{3}{8}a(2/3)^2 \\ &= \frac{1}{2}\left(\frac{1}{5} + \frac{2}{5}a\right) + \frac{3}{8}a(2/3)^2 \\ &= \frac{1}{10} + \frac{11}{30}a,\end{aligned}$$

   which we can solve to get $a = 1/3$.

   So our candidate π is $\pi_0 = 1/3$, $\pi_n = (1/3)(2/3)^n$, and in fact we can drop the special case for π_0.

   As a check, $\sum_{i=0}^{n} \pi_n = (1/3)\sum_{i=0}^{n}(2/3)^n = \frac{1/3}{1-2/3} = 1$.

2. Since $\mu_n = 1/\pi_n$, we have $\mu_n = 3(3/2)^n$.

3. In general, let $\mu_{k,n}$ be the expected time to reach n starting at k. Then $\mu_n = \mu_{n,n} = 1 + \frac{1}{8}\mu_{0,n} + \frac{1}{2}\mu_{n+1,n} + \frac{3}{8}\mu_{n-1,n} \geq 1 + \mu_{0,n}/8$. It follows that $\mu_{0,n} \leq 8\mu_n + 1 = 24(3/2)^n + 1 = O((3/2)^n)$.

   For the lower bound, observe that $\mu_n \leq \mu_{n,0} + \mu_{0,n}$. Since there is a $1/8$ chance of reaching 0 from any state, we have $\mu_{n,0} \leq 8$. It follows that $\mu_n \leq 8 + \mu_{0,n}$ or $\mu_{0,n} \geq \mu_n - 8 = \Omega((3/2)^n)$.

G.4.3 Yet another shuffling algorithm

Suppose we attempt to shuffle a deck of n cards by picking a card uniformly at random, and swapping it with the top card. Give the best bound you can on the mixing time for this process to reach a total variation distance of ϵ from the uniform distribution.

Solution

It's tempting to use the same coupling as for move-to-top (see §10.4.3). This would be that at each step we choose the same card to swap to the top position, which increases by at least one the number of cards that are in the same position in both decks. The problem is that at the next step, these two cards are most likely separated again, by being swapped with other cards in two different positions.

Instead, we will do something slightly more clever. Let Z_t be the number of cards in the same position at time t. If the top cards of both decks are equal, we swap both to the same position chosen uniformly at random. This has no effect on Z_t. If the top cards of both decks are not equal, we pick a card uniformly at random and swap it to the top in both decks. This increases Z_t by at least 1, unless we happen to pick cards that are already in the same position; so Z_t increases by at least 1 with probability $1 - Z_t/n$.

Let's summarize a state by an ordered pair (k, b) where $k = Z_t$ and b is 0 if the top cards are equal and 1 if they are not equal. Then we have a Markov chain where $(k, 0)$ goes to $(k, 1)$ with probability $\frac{n-k}{n}$ (and otherwise stays put); and $(k, 1)$ goes to $(k + 1, 0)$ (or higher) with probability $\frac{n-k}{n}$ and to $(k, 0)$ with probability $\frac{k}{n}$.

Starting from $(k, 0)$, we expect to wait $\frac{n}{n-k}$ steps on average to reach $(k, 1)$, at which point we move to $(k+1, 0)$ or back to $(k, 0)$ in one more step; we iterate through this process $\frac{n}{n-k}$ times on average before we are successful. This gives an expected number of steps to get from $(k, 0)$ to $(k + 1, 0)$ (or possibly a higher value) of $\frac{n}{n-k}\left(\frac{n}{n-k} + 1\right)\Big)$. Summing over k up to $n - 2$ (since once $k > n - 2$, we will in fact have $k = n$, since k can't be $n - 1$), we

get

$$\begin{aligned}
\mathrm{E}[\tau] &\leq \sum_{k=0}^{n-2} \frac{n}{n-k}\left(\frac{n}{n-k}+1\right) \\
&= \sum_{m=2}^{n}\left(\frac{n^2}{m^2}+\frac{n}{m}\right) \\
&\leq n^2\left(\frac{\pi^2}{6}-1\right)+n\ln n. \\
&= O(n^2).
\end{aligned}$$

So we expect the deck to mix in $O(n^2 \log(1/\epsilon))$ steps. (I don't know if this is the real bound; my guess is that it should be closer to $O(n \log n)$ as in all the other shuffling procedures.)

G.5 Assignment 5: due Thursday, 2011-04-07, at 23:59

G.5.1 A reversible chain

Consider a random walk on $\mathbb{Z}_m$, where $p_{i,i+1} = 2/3$ for all i and $p_{i,i-1} = 1/3$ for all i except $i = 0$. Is it possible to assign values to $p_{0,m-1}$ and $p_{0,0}$ to make this chain reversible, and if so, what stationary distribution do you get?

Solution

Suppose we can make this chain reversible, and let π be the resulting stationary distribution. From the detailed balance equations, we have $(2/3)\pi_i = (1/3)\pi_{i+1}$ or $\pi_{i+1} = 2\pi_i$ for $i = 0 \ldots m-2$. The solution to this recurrence is $\pi_i = 2^i\pi_0$, which gives $\pi_i = \frac{2^i}{2^m-1}$ when we set π_0 to get $\sum_i \pi_i = 1$.

Now solve $\pi_0 p_{0,m-1} = \pi_{m-1}p_{m-1,0}$ to get

$$\begin{aligned}
p_{0,m-1} &= \frac{\pi_{m-1}p_{m-1,0}}{\pi_0} \\
&= 2^{m-1}(2/3) \\
&= 2^m/3.
\end{aligned}$$

This is greater than 1 for $m > 1$, so except for the degenerate cases of $m = 1$ and $m = 2$, it's not possible to make the chain reversible.

G.5.2 Toggling bits

Consider the following Markov chain on an array of n bits $a[1], a[2], \ldots a[n]$. At each step, we choose a position i uniformly at random. We then change $A[i]$ to $\neg A[i]$ with probability $1/2$, provided $i = 1$ or $A[i-1] = 1$ (if neither condition holds hold, do nothing).[3]

1. What is the stationary distribution?

2. How quickly does it converge?

Solution

1. First let's show irreducibility. Starting from an arbitrary configuration, repeatedly switch the leftmost 0 to a 1 (this is always permitted by the transition rules); after at most n steps, we reach the all-1 configuration. Since we can repeat this process in reverse to get to any other configuration, we get that every configuration is reachable from every other configuration in at most $2n$ steps ($2n - 1$ if we are careful about handling the all-0 configuration separately).

   We also have that for any two adjacent configurations x and y, $p_{xy} = p_{yx} = \frac{1}{2n}$. So we have a reversible, irreducible, aperiodic (because there exists at least one self-loop) chain with a uniform stationary distribution $\pi_x = 2^{-n}$.

2. Here is a bound using the obvious coupling, where we choose the same position in X and Y and attempt to set it to the same value. To show this coalesces, given X_t and Y_t define Z_t to be the position of the rightmost 1 in the common prefix of X_t and Y_t, or 0 if there is no 1 in the common prefix of X_t and Y_t. Then Z_t increases by at least 1 if we attempt to set position $Z_t + 1$ to 1, which occurs with probability $\frac{1}{2n}$, and decreases by at most 1 if we attempt to set Z_t to 0, again with probability $\frac{1}{2n}$.

   It follows that Z_t reaches n no later than a ± 1 random walk on $0 \ldots n$ with reflecting barriers that takes a step every $1/n$ time units on average. The expected number of steps to reach n from the worst-case

[3] Motivation: Imagine each bit represents whether a node in some distributed system is inactive (0) or active (1), and you can only change your state if you have an active left neighbor to notify. Also imagine that there is an always-active *base station* at -1 (alternatively, imagine that this assumption makes the problem easier than the other natural arrangement where we put all the nodes in a ring).

starting position of 0 is exactly n^2. (Proof: model the random walk with a reflecting barrier at 0 by folding a random walk with absorbing barriers at $\pm n$ in half, then use the bound from §9.4.1.) We must then multiply this by n to get an expected n^3 steps in the original process. So the two copies coalesce in at most n^3 expected steps. My suspicion is one could improve this bound with a better analysis by using the bias toward increasing Z_t to get the expected time to coalesce down to $O(n^2)$, but I don't know any clean way to do this.

The path coupling version of this is that we look at two adjacent configurations X_t and Y_t, use the obvious coupling again, and see what happens to $\mathrm{E}\left[d(X_{t+1}, Y_{t+1}) \mid X_t, Y_t\right]$, where the distance is the number of transitions needed to convert X_t to Y_t or vice versa. If we pick the position i where X_t and Y_t differ, then we coalesce; this occurs with probability $1/n$. If we change the 1 to the left of i to a 0, then $d(X_{t+1}, Y_{t+1})$ rises to 3 (because to get from X_{t+1} to Y_{t+1}, we have to change position $i-1$ to 1, change position i, and then change position $i-1$ back to 0); this occurs with probability $1/2n$ if $i > 1$. But we can also get into trouble if we try to change position $i+1$; we can only make the change in one of X_t and Y_t, so we get $d(X_{t+1}, Y_{t+1}) = 2$ in this case, which occurs with probability $1/2n$ when $i < n$. Adding up all three cases gives a worst-case expected change of $-1/n + 2/2n + 1/2n = 1/2n > 0$. So unless we can do something more clever, path coupling won't help us here.

However, it is possible to get a bound using canonical paths, but the best bound I could get was not as good as the coupling bound. The basic idea is that we will change x into y one bit at a time (from left to right, say), so that we will go through a sequence of intermediate states of the form $y[1]y[2]\ldots y[i]x[i+1]x[i+2]\ldots x[n]$. But to change $x[i+1]$ to $y[i+1]$, we may also need to reach out with a tentacle of 1 bits from from the last 1 in the current prefix of y (and then retract it afterwards). Given a particular transition where we change a 0 to a 1, we can reconstruct the original x and y by specifying (a) which bit i at or after our current position we are trying to change; (b) which 1 bit before our current position is the last "real" 1 bit in y as opposed to something we are creating to reach out to position i; and (c) the values of $x[1]\ldots x[i-1]$ and $y[i+1]\ldots y[i]$. A similar argument applies to $1 \to 0$ transitions. So we are routing at most $n^2 2^{n-1}$ paths across each

transition, giving a bound on the congestion

$$\begin{aligned}\rho &\leq \left(\frac{1}{2^{-n}/2n}\right) n^2 2^{n-1} 2^{-2n} \\ &= n^3.\end{aligned}$$

The bound on τ_2 that follows from this is $8n^6$, which is pretty bad (although the constant could be improved by counting the (a) and (b) bits more carefully). As with the coupling argument, it may be that there is a less congested set of canonical paths that gives a better bound.

G.5.3 Spanning trees

Suppose you have a connected graph $G = (V, E)$ with n nodes and m edges. Consider the following Markov process. Each state H_t is a subgraph of G that is either a spanning tree or a spanning tree plus an additional edge. At each step, flip a fair coin. If it comes up heads, choose an edge e uniformly at random from E and let $H_{t+1} = H_t \cup \{e\}$ if H_t is a spanning tree and let $H_{t+1} = H_t \setminus \{e\}$ if H_t is not a spanning tree and $H_t \setminus \{e\}$ is connected. If it comes up tails and H_t is a spanning tree, let H_{t+1} be some other spanning tree, sampled uniformly at random. In all other cases, let $H_{t+1} = H_t$.

Let N be the number of states in this Markov chain.

1. What is the stationary distribution?

2. How quickly does it converge?

Solution

1. Since every transition has a matching reverse transition with the same transition probability, the chain is reversible with a uniform stationary distribution $\pi_H = 1/N$.

2. Here's a coupling that coalesces in at most $4m/3 + 2$ expected steps:

   (a) If X_t and Y_t are both trees, then send them to the same tree with probability 1/2; else let them both add edges independently (or we could have them add the same edge—it doesn't make any difference to the final result).

   (b) If only one of X_t and Y_t is a tree, with probability 1/2 scramble the tree while attempting to remove an edge from the non-tree,

and the rest of the time scramble the non-tree (which has no effect) while attempting to add an edge to the tree. Since the non-tree has at least three edges that can be removed, this puts $(X_{t+1}, Y_{t+1}$ in the two-tree case with probability at least $3/2m$.

(c) If neither X_t nor Y_t is a tree, attempt to remove an edge from both. Let S and T be the sets of edges that we can remove from X_t and Y_t, respectively, and let $k = \min(|S|, |T|) \geq 3$. Choose k edges from each of S and T and match them, so that if we remove one edge from each pair, we also remove the other edge. As in the previous case, this puts $(X_{t+1}, Y_{t+1}$ in the two-tree case with probability at least $3/2m$.

To show this coalesces, starting from an arbitrary state, we reach a two-tree state in at most $2m/3$ expected steps. After one more step, we either coalesce (with probability $1/2$) or restart from a new arbitrary state. This gives an expected coupling time of at most $2(2m/3 + 1) = 4m/3 + 2$ as claimed.

G.6 Assignment 6: due Monday, 2011-04-25, at 17:00

G.6.1 Sparse satisfying assignments to DNFs

Given a formula in disjunctive normal form, we'd like to estimate the number of satisfying assignments in which exactly w of the variables are true. Give a fully polynomial-time randomized approximation scheme for this problem.

Solution

Essentially, we're going to do the Karp-Luby covering trick [KL85] described in §11.4, but will tweak the probability distribution when we generate our samples so that we only get samples with weight w.

Let U be the set of assignment with weight w (there are exactly $\binom{n}{w}$ such assignments, where n is the number of variables). For each clause C_i, let $U_i = \{x \in U \mid C_i(x) = 1\}$. Now observe that:

1. We can compute $|U_i|$. Let $k_i = |C_i|$ be the number of variables in C_i and $k_i^+ = \left|C_i^+\right|$ the number of variables that appear in positive form in C_i. Then $|U_i| = \binom{n-k_i}{w-k_i^+}$ is the number of ways to make a total of w variables true using the remaining $n - k_i$ variables.

2. We can sample uniformly from U_i, by sampling a set of $w - k_i^+$ true variables not in C_i uniformly from all variables not in C_i.

3. We can use the values computed for $|U_i|$ to sample i proportionally to the size of $|U_i|$.

So now we sample pairs (i, x) with $x \in U_i$ uniformly at random by sampling i first, then sampling $x \in U_i$. As in the original algorithm, we then count (i, x) if and only if C_i is the leftmost clause for which $C_i(x) = 1$. The same argument that at least $1/m$ of the (i, x) pairs count applies, and so we get the same bounds as in the original algorithm.

G.6.2 Detecting duplicates

Algorithm G.1 attempts to detect duplicate values in an input array S of length n.

```
1 Initialize A[1...n] to ⊥
2 Choose a hash function h
3 for i ← 1...n do
4 |   x ← S[i]
5 |   if A[h(x)] = x then
6 |   |   return true
7 |   else
8 |   |   A[h(x)] ← x
9 return false
```

Algorithm G.1: Dubious duplicate detector

It's easy to see that Algorithm G.1 never returns **true** unless some value appears twice in S. But maybe it misses some duplicates it should find.

1. Suppose h is a random function. What is the worst-case probability that Algorithm G.1 returns **false** if S contains two copies of some value?

2. Is this worst-case probability affected if h is drawn instead from a 2-universal family of hash functions?

Solution

1. Suppose that $S[i] = S[j] = x$ for $i < j$. Then the algorithm will see x in $A[h(x)]$ on iteration j and return **true**, unless it is overwritten by some

value $S[k]$ with $i < k < j$. This occurs if $h(S[k]) = h(x)$, which occurs with probability exactly $1 - (1 - 1/n)^{j-i-1}$ if we consider all possible k. This quantity is maximized at $1 - (1 - 1/n)^{n-2} \approx 1 - (1 - 1/n)^2/e \approx 1 - (1 - 1/2n)/e$ when $i = 1$ and $j = n$.

2. As it happens, the algorithm can fail pretty badly if all we know is that h is 2-universal. What we can show is that the probability that some $S[k]$ with $i < j < k$ gets hashed to the same place as $x = S[i] = S[j]$ in the analysis above is at most $(j - i - 1)/n \leq (n - 2)/n = (1 - 2/n)$, since each $S[k]$ has at most a $1/n$ chance of colliding with x and the union bound applies. But it is possible to construct a 2-universal family for which we get exactly this probability in the worst case.

   Let $U = \{0 \dots n\}$, and define for each a in $\{0 \dots n - 1\}$ $h_a : U \to n$ by $h_a(n) = 0$ and $h_a(x) = (x + a) \bmod n$ for $x \neq n$. Then $H = \{h_a\}$ is 2-universal, since if $x \neq y$ and neither x nor y is n, $\Pr[h_a(x) = h_a(y)] = 0$, and if one of x or y is n, $\Pr[h_a(x) = h_a(y)] = 1/n$. But if we use this family in Algorithm G.1 with $S[1] = S[n] = n$ and $S[k] = k$ for $1 < k < n$, then there are $n - 2$ choices of a that put one of the middle values in $A[0]$.

G.6.3 Balanced Bloom filters

A clever algorithmist decides to solve the problem of Bloom filters filling up with ones by capping the number of ones at $m/2$. As in a standard Bloom filter, an element is inserted by writing ones to $A[h_1(x)], A[h_2(x)], \dots, A[h_k(x)]$; but after writing each one, if the number of one bits in the bit-vector is more than $m/2$, one of the ones in the vector (chosen uniformly at random) is changed back to a zero.

Because some of the ones associated with a particular element might be deleted, the membership test answers yes if at least 3/4 of the bits $A[h_1(x)] \dots A[h_k(x)]$ are ones.

To simplify the analysis, you may assume that the h_i are independent random functions. You may also assume that $(3/4)k$ is an integer.

1. Give an upper bound on the probability of a false positive when testing for a value x that has never been inserted.

2. Suppose that we insert x at some point, and then follow this insertion with a sequence of insertions of new, distinct values $y_1, y_2, \dots$. Assuming a worst-case state before inserting x, give asymptotic upper and

lower bounds on the expected number of insertions until a test for x fails.

Solution

1. The probability of a false positive is maximized when exactly half the bits in A are one. If x has never been inserted, each $A[h_i(x)]$ is equally likely to be zero or one. So $\Pr[\text{false positive for } x] = \Pr[S_k \geq (3/4)k]$ when S_k is a binomial random variable with parameters $1/2$ and k. Chernoff bounds give

$$\begin{aligned}\Pr[S_k \geq (3/4)k] &= \Pr[S_k \geq (3/2)\operatorname{E}[S_k]] \\ &\leq \left(\frac{e^{1/2}}{(3/2)^{3/2}}\right)^{k/2} \\ &\leq (0.94734)^k.\end{aligned}$$

   We can make this less than any fixed ϵ by setting $k \geq 20\ln(1/\epsilon)$ or thereabouts.

2. For false negatives, we need to look at how quickly the bits for x are eroded away. A minor complication is that the erosion may start even as we are setting $A[h_1(x)] \ldots A[h_k(x)]$.

   Let's consider a single bit $A[i]$ and look at how it changes after (a) setting $A[i] = 1$, and (b) setting some random $A[r] = 1$.

   In the first case, $A[i]$ will be 1 after the assignment unless it is set back to zero, which occurs with probability $\frac{1}{m/2+1}$. This distribution does not depend on the prior value of $A[i]$.

   In the second case, if $A[i]$ was previously 0, it becomes 1 with probability

$$\begin{aligned}\frac{1}{m}\left(1 - \frac{1}{m/2+1}\right) &= \frac{1}{m} \cdot \frac{m/2}{m/2+1} \\ &= \frac{1}{m+2}.\end{aligned}$$

   If it was previously 1, it becomes 0 with probability

$$\frac{1}{2} \cdot \frac{1}{m/2+1} = \frac{1}{m+2}.$$

So after the initial assignment, $A[i]$ just flips its value with probability $\frac{1}{m+2}$.

It is convenient to represent $A[i]$ as ± 1; let $X_i^t = -1$ if $A[i] = 0$ at time t, and 1 otherwise. Then X_i^t satisfies the recurrence

$$\begin{aligned}\mathrm{E}\left[X_i^{t+1} \;\middle|\; X_i^t\right] &= \frac{m+1}{m+2}X_i^t - \frac{1}{m+2}X_i^t \\ &= \frac{m}{m+2}X_i^t. \\ &= \left(1 - \frac{2}{m+2}\right)X_i^t.\end{aligned}$$

We can extend this to $\mathrm{E}\left[X_i^t \mid X_i^0\right] = \left(1 - \frac{2}{m+2}\right)^t X_i^0 \approx e^{-2t/(m+2)}X_i^0$.

Similarly, after setting $A[i] = 1$, we get $\mathrm{E}[X_i] = 1 - 2\frac{1}{m/2+1} = 1 - \frac{4}{2m+1} = 1 - o(1)$.

Let $S^t = \sum_{i=1}^k X_{h_i(x)}^t$. Let 0 be the time at which we finish inserting x. Then each for each i we have

$$1 - o(1)e^{-2k/(m+2)} \le \mathrm{E}\left[X_{h_i(x)}^0\right] \le 1 - o(1),$$

from which it follows that

$$k(1 - o(1))e^{-2k/(m+2)} \le \mathrm{E}\left[S^0\right] \le 1 - o(1)$$

and in general that

$$k(1 - o(1))e^{-2(k+t)/(m+2)} \le \mathrm{E}\left[S^t\right] \le 1 - o(1)e^{-2t/(m+2)}.$$

So for any fixed $0 < \epsilon < 1$ and sufficiently large m, we will have $\mathrm{E}\left[S^t\right] = \epsilon k$ for some t' where $t \le t' \le k + t$ and $t = \Theta(m \ln(1/\epsilon))$.

We are now looking for the time at which S^t drops below $k/2$ (the $k/2$ is because we are working with ± 1 variables). We will bound when this time occurs using Markov's inequality.

Let's look for the largest time t with $\mathrm{E}\left[S^t\right] \ge (3/4)k$. Then $\mathrm{E}\left[k - S^t\right] \le k/4$ and $\Pr\left[k - S^t \ge k/2\right] \le 1/2$. It follows that after $\Theta(m) - k$ operations, x is still visible with probability $1/2$, which implies that the expected time at which it stops being visible is at least $(\Omega(m) - k)/2$. To get the expected number of insert operations, we divide by k, to get $\Omega(m/k)$.

For the upper bound, apply the same reasoning to the first time at which $\mathrm{E}[S^t] \leq k/4$. This occurs at time $O(m)$ at the latest (with a different constant), so after $O(m)$ steps there is at most a $1/2$ probability that $S^t \geq k/2$. If S^t is still greater than $k/2$ at this point, try again using the same analysis; this gives us the usual geometric series argument that $\mathrm{E}[t] = O(m)$. Again, we have to divide by k to get the number of insert operations, so we get $O(m/k)$ in this case.

Combining these bounds, we have that x disappears after $\Theta(m/k)$ insertions on average. This seems like about what we would expect.

G.7 Final exam

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

G.7.1 Leader election

Suppose we have n processes and we want to elect a leader. At each round, each process flips a coin, and drops out if the coin comes up tails. We win if in some round there is exactly one process left.

Let $T(n)$ be the probability that this event eventually occurs starting with n processes. For small n, we have $T(0) = 0$ and $T(1) = 1$. Show that there is a constant $c > 0$ such that $T(n) \geq c$ for all $n > 1$.

Solution

Let's suppose that there is some such c. We will necessarily have $c \leq 1 = T(1)$, so the induction hypothesis will hold in the base case $n = 1$.

For $n \geq 2$, compute

$$\begin{aligned} T(n) &= \sum_{k=0}^{n} 2^{-n} \binom{n}{k} T(k) \\ &= 2^{-n}T(n) + 2^{-n}nT(1) + \sum_{k=2}^{n-1} 2^{-n} \binom{n}{k} T(k) \\ &\geq 2^{-n}T(n) + 2^{-n}n + 2^{-n}(2^n - n - 2)c. \end{aligned}$$

Solve for $T(n)$ to get

$$\begin{aligned} T(n) &\geq \frac{n + (2^n - n - 2)c}{2^n - 1} \\ &= c\left(\frac{2^n - n - 2 + n/c}{2^n - 1}\right). \end{aligned}$$

This will be greater than or equal to c if $2^n - n - 2 + n/c \geq 2^n - 1$ or $n/c \geq n + 1$, which holds if $c \leq \frac{n}{n+1}$. The worst case is $n = 2$ giving $c = 2/3$.

Valiant and Vazirani [VV86] used this approach to reduce solving general instances of SAT to solving instances of SAT with unique solutions; they prove essentially the result given above (which shows that fixing variables in a SAT formula is likely to produce a SAT formula with a unique solution at some point) with a slightly worse constant.

G.7.2 Two-coloring an even cycle

Here is a not-very-efficient algorithm for 2-coloring an even cycle. Every node starts out red or blue. At each step, we pick one of the n nodes uniformly at random, and change its color if it has the same color as at least one of its neighbors. We continue until no node has the same color as either of its neighbors.

Suppose that in the initial state there are exactly two monochromatic edges. What is the worst-case expected number of steps until there are no monochromatic edges?

Solution

Suppose we have a monochromatic edge surrounded by non-monochrome edges, e.g. $RBRRBR$. If we pick one of the endpoints of the edge (say the left endpoint in this case), then the monochromatic edge shifts in the direction of that endpoint: $RBBRBRB$. Picking any node not incident to a monochromatic edge has no effect, so in this case there is no way to increase the number of monochromatic edges.

It may also be that we have two adjacent monochromatic edges: $BRRRB$. Now if we happen to pick the node in the middle, we end up with no monochromatic edges ($BRBRB$) and the process terminates. If on the other hand we pick one of the nodes on the outside, then the monochromatic edges move away from each other.

We can thus model the process with 2 monochromatic edges as a random walk, where the difference between the leftmost nodes of the edges (mod

n) increases or decreases with equal probability $2/n$ except if the distance is 1 or -1; in this last case, the distance increases (going to 2 or -2) with probability $2/n$, but decreases only with probability $1/n$. We want to know when this process hits 0 (or n).

Imagine a random walk starting from position k with absorbing barriers at 1 and $n-1$. This reaches 1 or $n-1$ after $(k-1)(n-1-k)$ steps on average, which translates into $(n/4)(k-1)(n-1-k)$ steps of our original process if we take into account that we only move with probability $4/n$ per time unit. This time is maximized by setting $k = n/2$, which gives $(n/4)(n/2-1)^2 = n^3/16 - n^2/4 + n/4$ expected time units to reach 1 or $n-1$ for the first time.

At 1 or $n-1$, we wait an addition $n/3$ steps on average; then with probability $1/3$ the process finishes and with probability $2/3$ we start over from position 2 or $n-2$; in the latter case, we run $(n/4)(n-3)+n/3$ time units on average before we may finish again. On average, it takes 3 attempts to finish. Each attempt incurs the expected $n/3$ cost before taking a step, and all but the last attempt incur the expected $(n/4)(n-3)$ additional steps for the random walk. So the last phase of the process the process adds $(1/2)n(n-3)+n = (1/2)n^2 - (5/4)n$ steps.

Adding up all of the costs gives $n^3/16 - n^2/4 + n/4 + n/3 + (1/2)n^2 - (5/4)n = \frac{1}{16}n^3 + \frac{1}{4}n^2 - \frac{2}{3}n$ steps.

G.7.3 Finding the maximum

```
1 Randomly permute A
2 m ← −∞
3 for i ← 1 . . . n do
4     if A[i] > m then
5         m ← A[i]
6 return m
```

Algorithm G.2: Randomized max-finding algorithm

Suppose that we run Algorithm G.2 on an array with n elements, all of which are distinct. What is the expected number of times Line 5 is executed as a function of n?

Solution

Let X_i be the indicator variable for the event that Line 5 is executed on the i-th pass through the loop. This will occur if $A[i]$ is greater than $A[j]$ for all $j < i$, which occurs with probability exactly $1/i$ (given that A has been permuted randomly). So the expected number of calls to Line 5 is $\sum i = 1^n \operatorname{E}[X_i] = \sum_{i=1}^{n} \frac{1}{i} = H_n$.

G.7.4 Random graph coloring

Let G be a random d-regular graph on n vertices, that is, a graph drawn uniformly from the family of all n-vertex graph in which each vertex has exactly d neighbors. Color the vertices of G red or blue independently at random.

1. What is the expected number of monochromatic edges in G?

2. Show that the actual number of monochromatic edges is tightly concentrated around its expectation.

Solution

The fact that G is itself a random graph is a red herring; all we really need to know is that it's d-regular.

1. Because G has exactly $dn/2$ edges, and each edge has probability $1/2$ of being monochromatic, the expected number of monochromatic edges is $dn/4$.

2. This is a job for Azuma's inequality. Consider the vertex exposure martingale. Changing the color of any one vertex changes the number of monochromatic edges by at most d. So we have $\Pr[|X - \operatorname{E}[X]| \geq t] \leq 2\exp\left(-t^2/2\sum_{i=1}^{n} d^2\right) = 2e^{-t^2/2nd^2}$, which tells us that the deviation is likely to be not much more than $O(d\sqrt{n})$.

Appendix H

Sample assignments from Spring 2009

H.1 Final exam, Spring 2009

Write your answers in the blue book(s). Justify your answers. Work alone. Do not use any notes or books.

There are four problems on this exam, each worth 20 points, for a total of 80 points. You have approximately three hours to complete this exam.

H.1.1 Randomized mergesort (20 points)

Consider the following randomized version of the mergesort algorithm. We take an unsorted list of n elements and split it into two lists by flipping an independent fair coin for each element to decide which list to put it in. We then recursively sort the two lists, and merge the resulting sorted lists. The merge procedure involves repeatedly comparing the smallest element in each of the two lists and removing the smaller element found, until one of the lists is empty.

Compute the expected number of comparisons needed to perform this final merge. (You do not need to consider the cost of performing the recursive sorts.)

Solution

Color the elements in the final merged list red or blue based on which sublist they came from. The only elements that do not require a comparison to insert into the main list are those that are followed only by elements of the

same color; the expected number of such elements is equal to the expected length of the longest monochromatic suffix. By symmetry, this is the same as the expected longest monochromatic prefix, which is equal to the expected length of the longest sequence of identical coin-flips.

The probability of getting k identical coin-flips in a row followed by a different coin-flip is exactly 2^{-k}; the first coin-flip sets the color, the next $k-1$ must follow it (giving a factor of 2^{-k+1}, and the last must be the opposite color (giving an additional factor of 2^{-1}). For n identical coin-flips, there is a probability of 2^{-n+1}, since we don't need an extra coin-flip of the opposite color. So the expected length is $\sum_{k=1}^{n-1} k2^{-k} + n2^{-n+1} = \sum_{k=0}^{n} k2^{-k} + n2^{-n}$.

We can simplify the sum using generating functions. The sum $\sum_{k=0}^{n} 2^{-k}z^k$ is given by $\frac{1-(z/2)^{n+1}}{1-z/2}$. Taking the derivative with respect to z gives $\sum_{k=0}^{n} 2^{-k}kz^{k-1} = (1/2)\frac{1-(z/2)^{n+1}}{1-z/2}^2 + (1/2)\frac{(n+1)(z/2)^n}{1-z/2}$. At $z = 1$ this is $2(1 - 2^{-n-1}) - 2(n + 1)2^{-n} = 2 - (n + 2)2^{-n}$. Adding the second term gives $\mathrm{E}[X] = 2 - 2 \cdot 2^{-n} = 2 - 2^{-n+1}$.

Note that this counts the expected number of elements for which we do not have to do a comparison; with n elements total, this leaves $n - 2 + 2^{-n+1}$ comparisons on average.

H.1.2 A search problem (20 points)

Suppose you are searching a space by generating new instances of some problem from old ones. Each instance is either good or bad; if you generate a new instance from a good instance, the new instance is also good, and if you generate a new instance from a bad instance, the new instance is also bad.

Suppose that your start with X_0 good instances and Y_0 bad instances, and that at each step you choose one of the instances you already have uniformly at random to generate a new instance. What is the expected number of good instances you have after n steps?

Hint: Consider the sequence of values $\{X_t/(X_t + Y_t)\}$.

Solution

We can show that the suggested sequence is a martingale, by computing

$$\begin{aligned}
\mathrm{E}\left[\frac{X_{t+1}}{X_{t+1}+Y_{t+1}} \;\middle|\; X_t, Y_t\right] &= \frac{X_t}{X_t+Y_t}\cdot\frac{X_t+1}{X_t+Y_t+1} + \frac{Y_t}{X_t+Y_t}\cdot\frac{X_t}{X_t+Y_t+1} \\
&= \frac{X_t(X_t+1)+Y_tX_t}{(X_t+Y_t)(X_t+Y_t+1)} \\
&= \frac{X_t(X_t+Y_t+1)}{(X_t+Y_t)(X_t+Y_t+1)} \\
&= \frac{X_t}{X_t+Y_t}.
\end{aligned}$$

From the martingale property we have $\mathrm{E}\left[\frac{X_n}{X_n+Y_n}\right] = \frac{X_0}{X_0+Y_0}$. But $X_n + Y_n = X_0 + Y_0 + n$, a constant, so we can multiply both sides by this value to get $\mathrm{E}[X_n] = X_0\left(\frac{X_0+Y_0+n}{X_0+Y_0}\right)$.

H.1.3 Support your local police (20 points)

At one point I lived in a city whose local police department supported themselves in part by collecting fines for speeding tickets. A speeding ticket would cost 1 unit (approximately $100), and it was unpredictable how often one would get a speeding ticket. For a price of 2 units, it was possible to purchase a metal placard to go on your vehicle identifying yourself as a supporter of the police union, which (at least according to local legend) would eliminate any fines for subsequent speeding tickets, but which would not eliminate the cost of any previous speeding tickets.

Let us consider the question of when to purchase a placard as a problem in on-line algorithms. It is possible to achieve a strict[1] competitive ratio of 2 by purchasing a placard after the second ticket. If one receives fewer than 2 tickets, both the on-line and off-line algorithms pay the same amount, and at 2 or more tickets the on-line algorithm pays 4 while the off-line pays 2 (the off-line algorithm purchased the placard before receiving any tickets at all).

1. Show that no deterministic algorithm can achieve a lower (strict) competitive ratio.

2. Show that a randomized algorithm can do so, against an oblivious adversary.

[1] I.e., with no additive constant.

Solution

1. Any deterministic algorithm essentially just chooses some fixed number m of tickets to collect before buying the placard. Let n be the actual number of tickets issued. For $m = 0$, the competitive ratio is infinite when $n = 0$. For $m = 1$, the competitive ratio is 3 when $n = 1$. For $m > 2$, the competitive ratio is $(m + 2)/2 > 2$ when $n = m$. So $m = 2$ is the optimal choice.

2. Consider the following algorithm: with probability p, we purchase a placard after 1 ticket, and with probability $q = 1 - p$, we purchase a placard after 2 tickets. This gives a competitive ratio of 1 for $n = 0$, $1 + 2p$ for $n = 1$, and $(3p + 4q)/2 = (4 - p)/2 = 2 - p/2$ for $n \geq 2$. There is a clearly a trade-off between the two ratios $1 + 2p$ and $2 - p/2$. The break-even point is when they are equal, at $p = 2/5$. This gives a competitive ratio of $1 + 2p = 9/5$, which is less than 2.

H.1.4 Overloaded machines (20 points)

Suppose n^2 jobs are assigned to n machines with each job choosing a machine independently and uniformly at random. Let the load on a machine be the number of jobs assigned to it. Show that for any fixed $\delta > 0$ and sufficiently large n, there is a constant $c < 1$ such that the maximum load exceeds $(1 + \delta)n$ with probability at most nc^n.

Solution

This is a job for Chernoff bounds. For any particular machine, the load S is a sum of independent indicator variables and the mean load is $\mu = n$. So we have

$$\Pr[S \geq (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^n.$$

Observe that $e^\delta/(1+\delta)^{1+\delta} < 1$ for $\delta > 0$. One proof of this fact is to take the log to get $\delta-(1+\delta)\log(1+\delta)$, which equals 0 at $\delta = 0$, and then show that the logarithm is decreasing by showing that $\frac{d}{d\delta} \cdots = 1 - \frac{1+\delta}{1+\delta} - \log(1 + \delta) = -\log(1 + \delta) < 0$ for all $\delta > 0$.

So we can let $c = e^\delta/(1 + \delta)^{1+\delta}$ to get a bound of c^n on the probability that any particular machine is overloaded and a bound of nc^n (from the union bound) on the probability that any of the machines is overloaded.

Appendix I

Probabilistic recurrences

Randomized algorithms often produce recurrences with ugly sums embedded inside them (see, for example, (1.3.1)). We'd like to have tools for pulling at least asymptotic bounds out of these recurrences without having to deal with the sums. This isn't always possible, but for certain simple recurrences we can make it work.

I.1 Recurrences with constant cost functions

Let us consider probabilistic recurrences of the form $T(n) = 1 + T(n - X_n)$, where X_n is a random variable with $0 < X_n \leq n$ and $T(0) = 0$. We assume we can compute a lower bound on $\mathrm{E}[X_n]$ for each n, and we want to translate this lower bound into an upper bound on $\mathrm{E}[T(n)]$.

I.2 Examples

- How long does it take to get our first heads if we repeatedly flip a coin that comes up heads with probability p? Even though we probably already know the answer to this, We can solve it by solving the recurrence $T(1) = 1 + T(1 - X_1), T(0) = 0$, where $\mathrm{E}[X_1] = p$.
- **Hoare's FIND** [Hoa61b], often called **QuickSelect**, is an algorithm for finding the k-th smallest element of an unsorted array. It works like QuickSort, only after partitioning the array around a random pivot we throw away the part that doesn't contain our target and recurse only on the surviving piece. How many rounds of this must we do? Here $\mathrm{E}[X_n]$ is more complicated, since after splitting our array of size

n into piles of size n' and $n - n' - 1$, we have to pick one or the other (or possibly just the pivot alone) based on the value of k.

- Suppose we start with n biased coins that each come up heads with probability p. In each round, we flip all the coins and throw away the ones that come up tails. How many rounds does it take to get rid of all of the coins? (This essentially tells us how tall a skip list [Pug90] can get.) Here we have $\mathrm{E}[X_n] = (1-p)n$.

- In the **coupon collector problem**, we sample from $1 \ldots n$ with replacement until we see ever value at least once. We can model this by a recurrence in which $T(k)$ is the time to get all the coupons given there are k left that we haven't seen. Here X_n is 1 with probability k/n and 0 with probability $(n-k)/n$, giving $\mathrm{E}[X_n] = k/n$.

- Let's play **Chutes and Ladders** without the chutes and ladders. We start at location n, and whenever it's our turn, we roll a fair six-sided die X and move to $n - X$ unless this value is negative, in which case we stay put until the next turn. How many turns does it take to get to 0?

I.3 The Karp-Upfal-Wigderson bound

This is a bound on the expected number of rounds to finish a process where we start with a problem instance of size n, and after one round of work we get a new problem instance of size $n - X_n$, where X_n is a random variable whose distribution depends on n. It was original described in a paper by Karp, Upfal, and Wigderson on analyzing parallel search algorithms [KUW88]. The bound applies when $\mathrm{E}[X_n]$ is bounded below by a non-decreasing function $\mu(n)$.

Lemma I.3.1. *Let a be a constant, let $T(n) = 1 + T(n - X_n)$, where for each n, X_n is an integer-valued random variable satisfying $0 \le X_n \le n - a$ and let $T(a) = 0$. Let $\mathrm{E}[X_n] \ge \mu(n)$ for all $n > a$, where μ is a positive non-decreasing function of n. Then*

$$\mathrm{E}[T(n)] \le \int_a^n \frac{1}{\mu(t)}\, dt. \tag{I.3.1}$$

To get an intuition for why this works, imagine that X_n is the speed at which we drop from n, expressed in units per round. Traveling at this speed, it takes $1/X_n$ rounds to cross from $k+1$ to k for any such interval

we pass. From the point of view of the interval $[k, k+1]$, we don't know which n we are going to start from before we cross it, but we do know that for any $n \geq k+1$ we start from, our speed will be at least $\mu(n) \geq \mu(k+1)$ on average. So the time it takes will be at most $\int_k^{k+1} \frac{1}{\mu(t)}\,dt$ on average, and the total time is obtained by summing all of these intervals.

Of course, this intuition is not even close to a real proof (among other things, there may be a very dangerous confusion in there between $1/\operatorname{E}[X_n]$ and $\operatorname{E}[1/X_n]$), so we will give a real proof as well.

Proof of Lemma I.3.1. This is essentially the same proof as in Motwani and Raghavan [MR95], but we add some extra detail to allow for the possibility that $X_n = 0$.

Let $p = \Pr[X_n = 0]$, $q = 1 - p = \Pr[X_n \neq 0]$. Note we have $q > 0$ because otherwise $\operatorname{E}[X_n] = 0 < \mu(n)$. Then we have

$$\begin{aligned}\operatorname{E}[T(n)] &= 1 + \operatorname{E}[T(n - X_n)] \\ &= 1 + p\operatorname{E}[T(n - X_n) \mid X_n = 0] + q\operatorname{E}[T(n - X_n) \mid X_n \neq 0] \\ &= 1 + p\operatorname{E}[T(n)] + q\operatorname{E}[T(n - X_n) \mid X_n \neq 0].\end{aligned}$$

Now we have $\operatorname{E}[T(n)]$ on both sides, which we don't like very much. So we collect it on the left-hand side:

$$(1-p)\operatorname{E}[T(n)] = 1 + q\operatorname{E}[T(n - X_n) \mid X_n \neq 0],$$

divide both sides by $q = 1 - p$, and apply the induction hypothesis:

$$\begin{aligned}\operatorname{E}[T(n)] &= 1/q + \operatorname{E}[T(n - X_n) \mid X_n \neq 0] \\ &= 1/q + \operatorname{E}[\operatorname{E}[T(n - X_n) \mid X_n] \mid X_n \neq 0] \\ &\leq 1/q + \operatorname{E}\left[\int_a^{n-X_n} \frac{1}{\mu(t)}\,dt \;\middle|\; X_n \neq 0\right] \\ &= 1/q + \operatorname{E}\left[\int_a^{n} \frac{1}{\mu(t)}\,dt - \int_{n-X_n}^{n} \frac{1}{\mu(t)}\,dt \;\middle|\; X_n \neq 0\right] \\ &\leq 1/q + \int_a^{n} \frac{1}{\mu(t)}\,dt - \operatorname{E}\left[\frac{X_n}{\mu(n)} \;\middle|\; X_n \neq 0\right] \\ &\leq 1/q + \int_a^{n} \frac{1}{\mu(t)}\,dt - \frac{\operatorname{E}[X_n \mid X_n \neq 0]}{\mu(n)}.\end{aligned}$$

The second-to-last step uses the fact that $\mu(t) \leq \mu(n)$ for $t \leq n$.

It may seem like we don't know what $\operatorname{E}[X_n \mid X_n \neq 0]$ is. But we know that $X_n \geq 0$, so we have $\operatorname{E}[X_n] = p\operatorname{E}[X_n \mid X_n = 0] + q\operatorname{E}[X_n \mid X_n \neq 0] =$

$q \operatorname{E}[X_n \mid X_n \neq 0]$. So we can solve for $\operatorname{E}[X_n \mid X_n \neq 0] = E[X_n]/q$. So let's plug this in:

$$\begin{aligned}\operatorname{E}[T(n)] &\leq 1/q + \int_a^n \frac{1}{\mu(t)}\,dt - \frac{\operatorname{E}[X_n]/q}{\mu(n)} \\ &\leq 1/q + \int_a^n \frac{1}{\mu(t)}\,dt - 1/q \\ &= \int_a^n \frac{1}{\mu(t)}\,dt.\end{aligned}$$

This concludes the proof. □

Now we just need to find some applications.

I.3.1 Waiting for heads

For the recurrence $T(1) = 1 + T(1 - X_1)$ with $\operatorname{E}[X_1] = p$, we set $\mu(n) = p$ and get $\operatorname{E}[T(1)] \leq \int_0^1 \frac{1}{p}\,dt = \frac{1}{p}$, which happens to be exactly the right answer.

I.3.2 Quickselect

In Quickselect, we pick a random pivot and split the original array of size n into three piles of size m (less than the pivot), 1 (the pivot itself), and $n - m - 1$ (greater than the pivot). We then figure out which of the three piles contains the k-th smallest element (depend on how k compares to $m - 1$) and recurse, stopping when we hit a pile with 1 element. It's easiest to analyze this by assuming that we recurse in the largest of the three piles, i.e., that our recurrence is $T(n) = 1 + \max(T(m), T(n - m - 1))$, where m is uniform in $0 \ldots n - 1$. The exact value of $\operatorname{E}[\max(m, n - m - 1)]$ is a little messy to compute (among other things, it depends on whether n is odd or even), but it's not hard to see that it's always less than $(3/4)n$. So letting $\mu(n) = n/4$, we get

$$\operatorname{E}[T(n)] \leq \int_1^n \frac{1}{t/4}\,dt = 4 \ln n.$$

I.3.3 Tossing coins

Here we have $\operatorname{E}[X_n] = (1 - p)n$. If we let $\mu(n) = (1 - p)n$ and plug into the formula without thinking about it too much, we get

$$\operatorname{E}[T(n)] \leq \int_0^n \frac{1}{(1-p)t}\,dt = \frac{1}{1-p}(\ln n - \ln 0).$$

That $\ln 0$ is trouble. We can fix it by making $\mu(n) = (1-p)\lceil n \rceil$, to get

$$\begin{aligned} \mathrm{E}\left[T(n)\right] &\le \int_{0+}^{n} \frac{1}{(1-p)\lceil t \rceil}\, dt \\ &= \frac{1}{1-p} \sum_{k=1}^{n} \frac{1}{k} \\ &= \frac{H_n}{1-p}. \end{aligned}$$

I.3.4 Coupon collector

Now that we know how to avoid dividing by zero, this is easy and fun. Let $\mu(x) = \lceil x \rceil / n$, then we have

$$\begin{aligned} \mathrm{E}\left[T(n)\right] &\le \int_{0+}^{n} \frac{n}{\lceil t \rceil}\, dt \\ &= \sum_{k=1}^{n} \frac{n}{k} \\ &= nH_n. \end{aligned}$$

As it happens, this is the exact answer for this case. This will happen whenever X is always a 0–1 variable[1] and we define $\mu(x) = \mathrm{E}\left[X \mid n = \lceil x \rceil\right]$, which can be seen by spending far too much time thinking about the precise sources of error in the inequalities in the proof.

I.3.5 Chutes and ladders

Let $\mu(n)$ be the expected drop from position n. We have to be a little bit careful about small n, but we can compute that in general $\mu(n) = \frac{1}{6}\sum_{i=0}^{\min(n,6)} i$. For fractional values x we will set $\mu(x) = \mu(\lceil x \rceil)$ as before.

[1] A **0–1 random variable** is also called a **Bernoulli random variable**, but 0–1 is shorter to type and more informative. Even more confusing, the underlying experiment that gives rise to a Bernoulli random variable goes by the entirely different name of a **Poisson trial**. Jacob Bernoulli and Siméon-Denis Poisson were great mathematicians, but there are probably better ways to remember them.

Then we have

$$\begin{aligned}\mathrm{E}\left[T(n)\right] &\leq \int_{0+}^{n} \frac{1}{\mu(t)}\,dt \\ &= \sum_{k=1}^{n} \frac{1}{\mu(k)}\end{aligned}$$

We can summarize the values in the following table:

n	$\mu(n)$	$1/\mu(n)$	$\sum 1/\mu(k)$
1	1/6	6	6
2	1/2	2	8
3	1	1	9
4	5/3	3/5	48/5
5	5/2	2/5	10
≥ 6	7/2	2/7	$10 + (2/7)(n-5) = (2/7)n + 65/7$

This is a slight overestimate; for example, we can calculate by hand that the expected waiting time for $n = 2$ is 6 and for $n = 3$ that it is $20/3 = 6 + 2/3$.

We can also consider the generalized version of the game where we start at n and drop by $1 \ldots n$ each turn as long as the drop wouldn't take us below 0. Now the expected drop from position k is $k(k+1)/2n$, and so we can apply the formula to get

$$\mathrm{E}\left[T(n)\right] \leq \sum_{k=1}^{n} \frac{2n}{k(k+1)}.$$

The sum of $\frac{1}{k(k+1)}$ when k goes from 1 to n happens to have a very nice value; it's exactly $\frac{n}{n+1} = 1 + \frac{1}{n+1}$.[2] So in this case we can rewrite the bound as $2n \cdot \frac{n}{n+1} = \frac{2n^2}{n+1}$.

I.4 High-probability bounds

So far we have only considered bounds on the expected value of $T(n)$. Suppose we want to show that $T(n)$ is in fact small with high probability,

[2]Proof: Trivially true for $n = 0$; for larger n, compute $\sum_{k=1}^{n} \frac{1}{k(k+1)} \sum_{k=1}^{n-1} \frac{1}{k(k+1)} + \frac{1}{n(n+1)} = \frac{n-1}{n} + \frac{1}{n(n+1)} = \frac{(n-1)(n+1)-1}{n(n+1)} = \frac{n^2}{n(n+1)} = n/(n+1)$.

i.e., a statement of the form $\Pr[T(n) \geq t] \leq \epsilon$. There are two natural ways to do this: we can repeatedly apply Markov's inequality to the expectation bound, or we can attempt to analyze the recurrence in more detail. The first method tends to give weaker bounds but it's easier.

I.4.1 High-probability bounds from expectation bounds

Given $\mathrm{E}[T(n)] \leq m$, we have $\Pr[T(n) \geq \alpha m] \leq \alpha^{-1}$. This does not give a very good bound on probability; if we want to show $\Pr[T(n) \geq t] \leq n^{-c}$ for some constant c (a typical high-probability bound), we need $t \geq n^c m$. But we can get a better bound if m bounds the expected time starting from any reachable state, as is the case for the class of problems we have been considering.

The idea is that if $T(n)$ exceeds αm, we restart the analysis and argue that $\Pr[T(n) \geq 2\alpha m \mid T(n) \geq \alpha m] \leq \alpha^{-1}$, from which it follows that $\Pr[T(n) \geq 2\alpha m] \leq \alpha^{-2}$. In general, for any non-negative integer k, we have $\Pr[T(n) \geq k\alpha m] \leq \alpha^{-k}$. Now we just need to figure out how to pick α to minimize this quantity for fixed t.

Let $t = k\alpha m$. Then $k = t/\alpha m$ and we seek to minimize $\alpha^{-t/\alpha m}$. Taking the logarithm gives $-(t/m)(\ln \alpha)/\alpha$. The t/m factor is irrelevant to the minimization problem, so we are left with minimizing $-(\ln \alpha)/\alpha$. Taking the derivative gives $-\alpha^{-2} + \alpha^{-2} \ln \alpha$; this is zero when $\ln \alpha = 1$ or $\alpha = e$. (This popular constant shows up often in problems like this.) So we get $\Pr[T(n) \geq kem] \leq e^{-k}$, or, letting $k = \ln(1/\epsilon)$, $\Pr[T(n) \geq em \ln(1/\epsilon)] \leq \epsilon$.

So, for example, we can get an n^{-c} bound on the probability of running too long by setting our time bound to $em \ln(n^c) = cem \ln n = O(m \log n)$. We can't hope to do better than $O(m)$, so this bound is tight up to a log factor.

I.4.2 Detailed analysis of the recurrence

As Lance Fortnow has explained,[3] getting rid of log factors is what theoretical computer science is all about. So we'd like to do better than an $O(m \log n)$ bound if we can. In some cases this is not too hard.

Suppose for each n, $T(n) = 1 + T(X)$, where $\mathrm{E}[X] \leq \alpha n$ for a fixed constant α. Let $X_0 = n$, and let X_1, X_2, etc., be the sequence of sizes of the remaining problem at time 1, 2, etc. Then we have $\mathrm{E}[X_1] \leq \alpha n$ from our assumption. But we also have $\mathrm{E}[X_2] = \mathrm{E}[\mathrm{E}[X_2 \mid X_1]] \leq \mathrm{E}[\alpha X_1] = \alpha \mathrm{E}[X_1] \leq \alpha^2 n$, and by induction we can show that $\mathrm{E}[X_k] \leq \alpha^k n$ for all k.

[3] http://weblog.fortnow.com/2009/01/soda-and-me.html

Since X_k is integer-valued, $\mathrm{E}[X_k]$ is an upper bound on $\Pr[X_k > 0]$; we thus get $\Pr[T(n) \geq k] = \Pr[X_k > 0] \leq \alpha^k n$. We can solve for the value of k that makes this less than ϵ: $k = -\log(n/\epsilon)/\log\alpha = \log_{1/\alpha} n + \log_{1/\alpha}(1/\epsilon)$.

For comparison, the bound on the expectation of $T(n)$ from Lemma I.3.1 is $H(n)/(1-\alpha)$. This is actually pretty close to $\log_{1/\alpha} n$ when α is close to 1, and is not too bad even for smaller α. But the difference is that the dependence on $\log(1/\epsilon)$ is additive with the tighter analysis, so for fixed c we get $\Pr[T(n) \geq t] \leq n^{-c}$ at $t = O(\log n + \log n^c) = O(\log n)$ instead of $O(\log n \log n^c) = O(\log^2 n)$.

I.5 More general recurrences

We didn't do these, but if we had to, it might be worth looking at Roura's Improved Master Theorems [Rou01].

Bibliography

[AA11] Dan Alistarh and James Aspnes. Sub-logarithmic test-and-set against a weak adversary. In *Distributed Computing: 25th International Symposium, DISC 2011*, volume 6950 of *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, September 2011.

[AACH12] James Aspnes, Hagit Attiya, and Keren Censor-Hillel. Polylogarithmic concurrent data structures from monotone circuits. *Journal of the ACM*, 59(1):2:1–2:24, February 2012.

[AAG+10] Dan Alistarh, Hagit Attiya, Seth Gilbert, Andrei Giurgiu, and Rachid Guerraoui. Fast randomized test-and-set and renaming. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2010.

[AB07] Sanjeev Arora and Boaz Barak. Computational complexity: A modern approach. Unpublished draft available at https://theory.cs.princeton.edu/complexity/book.pdf, 2007.

[ABMRT96] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. Static dictionaries on AC^0 RAMs: Query time $\theta(\sqrt{\log n / \log \log n})$ is necessary and sufficient. In *FOCS*, pages 441–450, 1996.

[Abr88] Karl Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the 2nd annual ACM-SIAM symposium on Discrete algorithms*, pages 291–302, 1988.

[AC08] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM*, 55(5):20, October 2008.

[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[Ach03] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, June 2003.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Washington, DC, USA, 1978. IEEE Computer Society.

[AE19] James Aspnes and He Yang Er. Consensus with max registers. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:9, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[AF01] David Aldous and James Allen Fill. Reversible Markov chains and random walks on graphs. Unpublished manuscript, available at https://www.stat.berkeley.edu/~aldous/RWG/book.html, 2001.

[AH90] James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, September 1990.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1983. ACM.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160:781–793, 2004.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of

approximation problems. *Journal of the ACM*, 45(3):501–555, may 1998.

[Alo91] Noga Alon. A parallel algorithmic version of the local lemma. *Random Structures & Algorithms*, 2(4):367–378, 1991.

[AM18] Vigleik Angeltveit and Brendan D. McKay. $r(5,5) \leq 48$. *Journal of Graph Theory*, 89(1):5–13, 2018.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[AS92] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1992.

[AS07] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37, November 2007.

[Asp15] James Aspnes. Faster randomized consensus with an oblivious adversary. *Distributed Computing*, 28(1):21–29, February 2015.

[Asp20] James Aspnes. Notes on computational complexity theory. Unpublished manuscript, 2020.

[AVL62] G. M. Adelson-Velskii and E. M. Landis. An information organization algorithm. *Doklady Akademia Nauk SSSR*, 146:263–266, 1962.

[AW96] James Aspnes and Orli Waarts. Randomized consensus in $O(n \log^2 n)$ operations per processor. *SIAM Journal on Computing*, 25(5):1024–1044, October 1996.

[Azu67] Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19(3):357–367, 1967.

[Bab79] László Babai. Monte-carlo algorithms in graph isomorphism testing. Technical Report D.M.S. 79-10, Université de Montréal, 1979.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, October 1997.

[BCB12] Sébastian Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[BD92] Dave Bayer and Persi Diaconis. Trailing the dovetail shuffle to its lair. *Annals of Applied Probability*, 2(2):294–313, 1992.

[BD97] Russ Bubley and Martin Dyer. Path coupling: A technique for proving rapid mixing in markov chains. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 223–231. IEEE, 1997.

[Bec91] József Beck. An algorithmic approach to the lovász local lemma. i. *Random Structures & Algorithms*, 2(4):343–365, 1991.

[Bel57] Richard Earnest Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. *computational complexity*, 1(1):3–40, 1991.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[Bol01] Béla Bollobás. *Random Graphs*. Cambridge University Press, second edition, 2001.

[BR91] Gabriel Bracha and Ophir Rachman. Randomized consensus in expected $O(n^2 \log n)$ operations. In Sam Toueg, Paul G. Spirakis, and Lefteris M. Kirousis, editors, *Distributed Algorithms, 5th International Workshop*, volume 579 of *Lecture Notes in Computer Science*, pages 143–150, Delphi, Greece, 7–9 October 1991. Springer, 1992.

[Bro86] Andrei Z. Broder. How hard is to marry at random? (on the approximation of the permanent). In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, 28-30 May 1986, Berkeley, California, USA*, pages 50–58, 1986.

[Bro88] Andrei Z. Broder. Errata to "how hard is to marry at random? (on the approximation of the permanent)". In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*, page 551, 1988.

[CM03] Saar Cohen and Yossi Matias. Spectral Bloom filters. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 241–252, 2003.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[CMV13] Kai-Min Chung, Michael Mitzenmacher, and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing*, 9(1):897–945, 2013.

[Coh15] Edith Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *IEEE Trans. Knowl. Data Eng.*, 27(9):2320–2334, 2015.

[Coh21] Gil Cohen. Two-source dispersers for polylogarithmic entropy and improved Ramsey graphs. *SIAM Journal on Computing*, 50(3):STOC16–30–STOC16–67, 2021.

[CS00] Artur Czumaj and Christian Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 30–39, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[CW77] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM.

[Deu89] David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[Dev88] Luc Devroye. Applications of the theory of records in the study of random trees. *Acta Informatica*, 26(1-2):123–130, October 1988.

[DG00] Martin Dyer and Catherine Greenhill. On Markov chains for independent sets. *J. Algorithms*, 35:17–49, April 2000.

[DG03] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, page 1–12, New York, NY, USA, 1987. Association for Computing Machinery.

[DGM02] Martin Dyer, Catherine Greenhill, and Mike Molloy. Very rapid mixing of the Glauber dynamics for proper colorings on bounded-degree graphs. *Random Struct. Algorithms*, 20:98–114, January 2002.

[DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997.

[Din07] Irit Dinur. The pcp theorem by gap amplification. *Journal of the ACM (JACM)*, 54(3):12, 2007.

[Dir39] P. A. M. Dirac. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 35:416–418, 6 1939.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[DKM+94] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer Auf Der Heide, Hans Rohnert, and Robert E Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[Dum56] A. I. Dumey. Indexing for rapid random-access memory. *Computers and Automation*, 5(12):6–9, 1956.

[Dye03] Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 693–699, New York, NY, USA, 2003. ACM.

[EL75] Paul Erdős and Laszlo Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V. T. Sós, editors, *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*, pages 609–627. North-Holland, 1975.

[Epp16] David Eppstein. Cuckoo filter: Simplification and analysis. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 8:1–8:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[Erd45] P. Erdős. On a lemma of Littlewood and Offord. *Bulletin of the American Mathematical Society*, 51(12):898–902, 1945.

[Erd47] P. Erdős. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53:292–294, 1947.

[ES35] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[Eul68] M. L. Euler. Remarques sur un beau rapport entre les séries des pussances tant directes que réciproques. *Mémoires de l'Académie des Sciences de Berlin*, 17:83–106, 1768.

[FAKM14] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than Bloom. In Aruna Seneviratne, Christophe Diot, Jim Kurose, Augustin Chaintreau, and Luigi Rizzo, editors, *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, pages 75–88. ACM, 2014.

[FCAB00] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, third edition, 1968.

[Fel71] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, second edition, 1971.

[FFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[FGQ12] Xiequan Fan, Ion Grama, and Liu Quansheng. Hoeffding's inequality for supermartingales. Available as https://arxiv.org/abs/1109.4359, July 2012.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a Sparse Table with O(1) Worst Case Access Time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[FNM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[FPSS03] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. In Helmut Alt and Michel Habib, editors, *STACS 2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 271–282. Springer Berlin Heidelberg, 2003.

[FR75] Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Commun. ACM*, 18(3):165–172, 1975.

[Fre21] Casper Benjamin Freksen. An introduction to Johnson-Lindenstrauss transforms. *arXiv preprint arXiv:2103.00564*, 2021.

[GKP88] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1988.

[Goo83] Nelson Goodman. *Fact, Fiction, and Forecast*. Harvard University Press, 1983.

[GR93] Igal Galperin and Ronald L. Rivest. Scapegoat trees. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1993.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996. Available as https://arxiv.org/abs/quant-ph/9605043.

[GS78] Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 8–21. IEEE, 1978.

[GS92] G. R. Grimmet and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science Publications, second edition, 1992.

[GS01] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes*. Oxford University Press, third edition, 2001.

[Gur00] Venkatesen Guruswami. Rapidly mixing Markov chains: A comparison of techniques. Available at ftp://theory.lcs.mit.edu/pub/people/venkat/markov-survey.ps, 2000.

[GW94] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discret. Math.*, 7:656–666, November 1994.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[GW12] George Giakkoupis and Philipp Woelfel. On the time and space complexity of randomized test-and-set. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 19–28. ACM, 2012.

[Has70] W. K. Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[Hec21] Annika Heckel. Non-concentration of the chromatic number of a random graph. *Journal of the American Mathematical Society*, 34(1):245–260, January 2021.

[HH80] P. Hall and C.C. Heyde. *Martingale Limit Theory and Its Application*. Academic Press, 1980.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[Hoa61a] C. A. R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4:321, July 1961.

[Hoa61b] C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4:321–322, July 1961.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[Hol16] Jonathan Holmes. AI is already making inroads into journalism but could it win a Pulitzer? https://www.theguardian.com/media/2016/apr/03/artificla-intelligence-robot-reporter-pulitzer-prize, April 2016. Accessed September 18th, 2016.

[HST08] Maurice Herlihy, Nir Shavit, and Moran Tzafrir. Hopscotch hashing. In *Proceedings of the 22nd international symposium on Distributed Computing*, DISC '08, pages 350–364, Berlin, Heidelberg, 2008. Springer-Verlag.

[HWY22] Kun He, Chunyang Wang, and Yitong Yin. Sampling lovász local lemma for general constraint satisfaction solutions in near-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 147–158, 2022.

[HY04] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3:21–35, 2004.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[Jer95] Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures & Algorithms*, 7(2):157–165, 1995.

[JL84] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability (New Haven, Connecticut, 1982)*, number 26 in Contemporary Mathematics, pages 189–206. American Mathematical Society, 1984.

[JLR00] Svante Janson, Tomasz Łuczak, and Andrezej Ruciński. *Random Graphs*. John Wiley & Sons, 2000.

[Joh10] Daniel Johannsen. *Random Combinatorial Structures and Randomized Search Heuristics*. PhD thesis, Universität des Saarlands, 2010.

[JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

[Kar93] David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 21–30, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[KGV83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM, 2002.

[Kho10] Subhash Khot. On the unique games conjecture. In *2010 25th Annual IEEE Conference on Computational Complexity*, pages 99–121, 2010.

[KL85] Richard M. Karp and Michael Luby. Monte-carlo algorithms for the planar multiterminal network reliability problem. *J. Complexity*, 1(1):45–64, 1985.

[KM08] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. *Random Struct. Algorithms*, 33(2):187–218, 2008.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM.

[Kol33] A.N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer, 1933.

[KR99] V. S. Anil Kumar and H. Ramesh. Markovian coupling vs. conductance for the Jerrum-Sinclair chain. In *FOCS*, pages 241–252, 1999.

[KS76] J.G. Kemeny and J.L. Snell. *Finite Markov Chains: With a New Appendix "Generalization of a Fundamental Matrix"*. Undergraduate Texts in Mathematics. Springer, 1976.

[KSK76] John G. Kemeny, J. Laurie. Snell, and Anthony W. Knapp. *Denumerable Markov Chains*, volume 40 of *Graduate Texts in Mathematics*. Springer, 1976.

[KT75] Samuel Karlin and Howard M. Taylor. *A First Course in Stochastic Processes*. Academic Press, second edition, 1975.

[KUW88] Richard M. Karp, Eli Upfal, and Avi Wigderson. The complexity of parallel search. *Journal of Computer and System Sciences*, 36(2):225–253, 1988.

[LAA87] Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, pages 163–183, 1987.

[Len20] Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer International Publishing, Cham, 2020.

[Li80] Shuo-Yen Robert Li. A martingale approach to the study of occurrence of sequence patterns in repeated experiments. *Annals of Probability*, 8(6):1171–1176, 1980.

[Lin92] Torgny Lindvall. *Lectures on the Coupling Method*. Wiley, 1992.

[LPW09] David A. Levin, Yuval Peres, and Elizabeth L. Wimmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[LR85] T. L. Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, March 1985.

[LS20] Tor Lattimore and Csaba Szepsesvári. *Bandit Algorithms*. Cambridge University Press, 1st edition, September 2020.

[Lub85] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1985. ACM.

[LV97] Michael Luby and Eric Vigoda. Approximately counting up to four (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 682–687, New York, NY, USA, 1997. ACM.

[LV99] Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures & Algorithms*, 15(3–4):229–241, 1999.

[LW05] Michael Luby and Avi Wigderson. Pairwise independence and derandomization. *Foundations and Trends in Theoretical Computer Science*, 1(4):237–301, 2005.

[McC85] Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[McD89] Colin McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics, 1989: Invited Papers at the Twelfth British Combinatorial Conference*, pages 148–188, 1989.

[MH92] Colin McDiarmid and Ryan Hayward. Strong concentration for Quicksort. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete algorithms*, SODA '92, pages 414–421, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[ML86] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 149–159. Morgan Kaufmann, 1986.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998.

[Mos09] Robin A. Moser. A constructive proof of the Lovász local lemma. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 343–350, New York, NY, USA, 2009. ACM.

[MPW16] Mikhail Menshikov, Serguei Popov, and Andrew Wade. *Nonhomogeneous random walks: Lyapunov function methods for near-critical stochastic systems*, volume 209. Cambridge University Press, 2016.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MR97] Brendan D McKay and Stanisław P Radziszowski. Subgraph counting identities and Ramsey numbers. *Journal of Combinatorial Theory, Series B*, 69(2):193–209, 1997.

[MR98] Michael Molloy and Bruce Reed. Further algorithmic aspects of the local lemma. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 524–529, New York, NY, USA, 1998. ACM.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

[MT05] Ravi Montenegro and Prasad Tetali. Mathematical aspects of mixing times in markov chains. *Foundations and Trends in Theoretical Computer Science*, 1(3), 2005.

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57:11:1–11:15, February 2010.

[MU17] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, second edition, 2017.

[Mun11] Randall Munroe. Sports. https://xkcd.com/904/, 2011. Accessed September 18th, 2016.

[MWW07] Elchanan Mossel, Dror Weitz, and Nicholas Wormald. On the hardness of sampling independent sets beyond the tree threshold. Available as https://arxiv.org/abs/math/0701471, 2007.

[Pag01] Rasmus Pagh. On the cell probe complexity of membership and perfect hashing. In *STOC*, pages 425–432, 2001.

[Pag06] Rasmus Pagh. Cuckoo hashing for undergraduates. Available at https://www.it-c.dk/people/pagh/papers/cuckoo-undergrad.pdf, 2006.

[Pap91] Christos H. Papadimitriou. On selecting a satisfying truth assignment (extended abstract). In *32nd Annual Symposium on*

Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991, pages 163–169. IEEE Computer Society, 1991.

[PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 823–829. SIAM, 2005.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[PT12] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14, 2012.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[PWY20] Seth Pettie, Dingyu Wang, and Longhui Yin. Non-mergeable sketching for cardinality estimation. Available as https://arxiv.org/abs/2008.08739, 2020.

[Rab80] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. Syst. Sci.*, 37:130–143, October 1988.

[Rou01] Salvador Roura. Improved master theorems for divide-and-conquer recurrences. *J. ACM*, 48:170–205, March 2001.

[RS14] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014. Genetic and Evolutionary Computation.

[RT87] Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, December 1987.

[SA96] Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996. Available at https://people.ischool.berkeley.edu/~aragon/pubs/rst96.pdf.

[Sch35] Erwin Schrödinger. Die gegenwärtige Situation in der Quantenmechanik. *Naturwissenschaften*, 23(48):807–812, November 1935.

[She11] Irina Shevtsova. On the asymptotically exact constants in the Berry-Esseen-Katz inequality. *Theory of Probability & Its Applications*, 55(2):225–252, 2011. A preprint is available at https://arxiv.org/pdf/1111.6554.pdf.

[Sho97] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM journal on computing*, 26(5):1484–1509, 1997.

[SMJ+14] Attila Szolnoki, Mauro Mobilia, Luo-Luo Jiang, Bartosz Szczesny, Alastair M Rucklidge, and Matjaž Perc. Cyclic dominance in evolutionary games: a review. *Journal of the Royal Society Interface*, 11(100):20140735, 2014.

[Spu12] Francis Spufford. *Red Plenty*. Graywolf Press, 2012.

[Sri08] Aravind Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 611–620, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3–4):281–292, 1971.

[SS87] E. Shamir and J. Spencer. Sharp concentration of the chromatic number on random graphs $G_{n,p}$. *Combinatorica*, 7:121–129, January 1987.

[Str03] Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press, 2003.

[SW13] Dennis Stanton and Dennis White. *Constructive Combinatorics*. Undergraduate Texts in Mathematics. Springer, 2013.

[Tin14] Daniel Ting. Streamed approximate counting of distinct elements: beating optimal batch methods. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 442–451. ACM, 2014.

[Tod91] Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[Tof80] Tommaso Toffoli. Reversible computing. Technical Report MIT/LCS/TM-151, MIT Laboratory for Computer Science, 1980.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Val82] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.

[VB81] Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *STOC*, pages 263–277. ACM, 1981.

[Vit85] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[VN63] John Von Neumann. Various techniques used in connection with random digits. *John von Neumann, Collected Works*, 5:768–770, 1963.

[Vui80] Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980.

[VV86] Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[Wil04] David Bruce Wilson. Mixing times of lozenge tiling and card shuffling Markov chains. *Annals of Applied Probability*, 14(1):274–325, 2004.

[Wil06] Herbert S. Wilf. *generatingfunctionology*. A. K. Peters, third edition, 2006. Second edition is available on-line at https://www.math.penn.edu/~wilf/DownldGF.html.

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pages 222–227. IEEE, 1977.

[Zob70] Albert L. Zobrist. A new hashing method with application for game playing. Technical Report 88, University of Washington, Computer Sciences Department, April 1970.

[Zur03] Wojciech Hubert Zurek. Decoherence, einselection, and the quantum origins of the classical. *Rev. Mod. Phys.*, 75:715–775, May 2003.

Index

(r_1, r_2, p_1, p_2)-sensitive, 141
2-coloring, 245
3-way cut, 460
$G(n, p)$, 91
$\langle r(n), q(n), \rho \rangle$-**PCP** verifier, 262
d-ary cuckoo hashing, 127
k-CNF formula, 245
k-uniform hypergraph, 245
k-universal, 116
k-wise independence, 21
t_{mix}, 174
Φ, 203
ϵ-NNS, 141
ϵ-PLEB, 141
ϵ-nearest neighbor search, 141
ϵ-point location in equal balls, 141
ρ-close, 266
σ-algebra, 16, 43
    generated by a random variable, 43
τ_2, 201
$\mathcal{F}$-measurable, 43
#DNF, 215, 220
#P, 214
#SAT, 214
PCP theorem, 262
0–1 random variable, 487
2-universal, 115

absolute value, 279
adapted, 84, 162
adapted sequence of random variables, 153
adaptive adversary, 107, 289
additive drift, 230
Adleman's theorem, 255
adversary, 289
    adaptive, 107, 289
    oblivious, 107, 289
advice, 255
agreement, 290
Aldous-Fill manuscript, 183
algorith
    gossip, 225
algorithm
    Deutsch's, 284
    distributed, 289
    epidemic, 225
    evolutionary, 230
    UCB1, 95
algorithmic information theory, 114
almost Markov, 322
alphabet, 270
amplitude
    probability, 274
ancestor, 107
    common, 108
annealing schedule, 196
anti-concentration bound, 100
aperiodic, 176
approximation algorithm, 239
approximation ratio, 239
approximation scheme

fully polynomial-time, 218
fully polynomial-time randomized, 214
arithmetic coding, 217, 450
arm, 94
Atari 2600, 373
augmenting path, 212
average-case analysis, 2
AVL tree, 103
avoiding worst-case inputs, 13
axioms of probability, 16
Azuma's inequality, 80, 85
Azuma-Hoeffding inequality, 80

balanced, 102
balancing
set, 235
bandit
multi-armed, 94
Bernoulli random variable, 33, 487
Berry-Esseen theorem, 100
biased random walk, 160
binary search tree, 102
balanced, 102
binary tree, 102
binomial random variable, 33
bipartite graph, 211
bit-fixing routing, 78, 208
Bloom filter, 128, 472
counting, 133
spectral, 133
Bloomjoin, 132
Boole's inequality, 53
bound
union, 53
bounded difference property, 88
bowling, 429
bra, 277
bra-ket notation, 277
bracket, 277
branching process, 251
bubble sort, 364
Byzantine processes, 289

canonical paths, 204
cardinality estimation, 134
cartesian tree, 105
Catalan number, 103
causal coupling, 198
CCNOT, 282
central limit theorem, 99
chaining, 115
Chapman-Kolmogorov equation, 167
Chebyshev's inequality, 60
Cheeger constant, 203
Cheeger inequality, 203
chromatic number, 91
Chutes and Ladders, 484
circuit
quantum, 274, 279
random, 275
clause, 220, 239, 245
clique, 236
closed loop, 125
CLT, 99
CNOT, 281
code
error-correcting, 265
Walsh-Hadamard, 265
codeword, 265
coin
weak shared, 292
collision, 13, 114, 115
coloring
hypergrah, 245
common ancestor, 108
common subsequence, 457
complement, 16
completeness, 262
complex conjugate, 277

computation
randomized, 276
computation path, 255
computation tree, 255
concave, 55
concentration bound, 59
coupon collector problem, 185
conditional expectation, 7, 36
conditional probability, 21
conductance, 203
congestion, 206
conjecture
Unique Games, 272
conjugate
complex, 277
conjugate transpose, 277
conjunctive normal form, 239
consensus, 290
constraint, 270
constraint graph, 270
constraint hypergraph, 271
constraint satisfaction problem, 270
continuous random variable, 30
contraction, 25
controlled controlled NOT, 282
controlled NOT, 281
convex, 54
convex function, 54
Cook reduction, 214
count-duplicates, 134
count-min sketch, 136
countable additivity, 16
countable Markov chain, 166
counting Bloom filter, 133
coupling, 174
causal, 198
path, 189
Coupling Lemma, 174
coupling time, 184, 186
coupon collector problem, 48, 484
concentration bounds, 185
covariance, 35, 62
crash, 290
cryptographic hash function, 14
cryptographically secure pseudorandom generator, 254
cryptographically secure pseudorandom number generator, 9
Cryptographically-secure pseudorandomness., 9
cuckoo, 122
cuckoo filter, 131
cuckoo hash table, 122
cuckoo hashing, 122
d-ary, 127
cumulative distribution function, 30
curse of dimensionality, 141
cut
minimum, 25
cylinder set, 17

data stream, 133
dead cat bounce, 369
decodable
locally, 266
decoding
local, 266
decoherence, 279
decomposition
Doob, 86
dense, 211
density, 333
dependency graph, 246
depth
tree, 402
derandomization, 253
detailed balance equations, 178
dictionary, 114
discrepancy, 235
discrete probability space, 16

discrete random variable, 28, 31
discretization, 146
disjoint, 16
disjunctive normal form, 220
distance
    total variation, 171
distributed algorithm, 289
distributed algorithms, 14
distributed computing, 289
distribution, 30
    geometric, 47
    stationary, 171
DNF formula, 220
dominating set, 445, 452
Doob decomposition, 86
Doob decomposition theorem, 155
Doob martingale, 88, 257
Doob's martingale inequality, 163
double record, 416
dovetail shuffle, 189
drift
    additive, 230
    multiplicative, 232
    variable, 231
drift analysis, 230
drift process, 155
Dungeons and Dragons, 368
dynamic perfect hashing, 122
dynamics
    Glauber, 193

eigenvalue, 198
eigenvector, 198
einselection, 279
entropy, 217
epidemic, 225
epidemic algorithm, 225
equation
    Wald's, 36
Erdős-Szekeres theorem, 431, 458
error budget, 53
error-correcting code, 265
Euler-Mascheroni constant, 8
event, 15, 16
events
    independent, 19
evolutionary algorithm, 230
execution log, 250
expectation, 31
    conditional, 36
    conditioned on a σ-algebra, 43
    conditioned on a random variable, 38
    conditioned on an event, 37
    iterated
      law of, 41
    linearity of, 32
expected time, 2
expected value, 2, 31
expected worst-case, 1
exponential generating function, 68

false positive, 128
filter
    cuckoo, 131
filtration, 44, 153
financial analysis, 368
fingerprint, 14, 131
fingerprinting, 14
finite Markov chain, 166
first passage time, 177
Flajolet-Martin sketch, 134, 433
FPRAS, 214
fractional solution, 239, 240
Frobenius problem, 176
fully polynomial-time approximation scheme, 218
fully polynomial-time randomized approximation scheme, 214
function

Lyapunov, 225
measurable, 28
potential, 226
futile word search puzzle, 443

game
unique, 272
gap, 268
gap-amplifying reduction, 269
gap-preserving reduction, 269
gap-reducing reduction, 269
gate
Toffoli, 282
generated by, 43
generating function
exponential, 68
probability, 33, 56
geometric distribution, 47
geometric random variable, 47
Glauber dynamics, 193
GNI, 263
gossip algorithm, 225
graph
bipartite, 211
random, 29, 91
GRAPH NON-ISOMORPHISM, 263
Grover diffusion operator, 286

Hadamard operator, 281
handwaving, 9, 250
harmonic number, 8
hash function, 13, 115
perfect, 121
tabulation, 119
universal, 115
hash table, 13, 114
cuckoo, 122
hashing, 13
cuckoo, 122
dynamic perfect, 122
hopscotch, 128
locality-sensitive, 140
Zobrist, 119
heap, 105
heavy hitter, 136
high probability, 51
high-probability bound, 1
hitting time, 224
Hoare's FIND, 49, 483
Hoeffding's inequality, 80
Hoeffding's lemma, 80
homogeneous, 166
hopscotch hashing, 128
hypercube network, 77, 90
hyperedge, 245, 271
hypergraph, 245
k-uniform, 245
HyperLogLog, 135

inclusion-exclusion formula, 19
independence, 19
of events, 19
of sets of events, 21
pairwise, 21
independent, 19, 21
independent events, 19
independent random variables, 30
independent set, 236
index, 114
indicator random variable, 29
indicator variable, 7
inequality
Azuma's, 80, 85
Azuma-Hoeffding, 80
Boole's, 53
Chebyshev's, 60
Cheeger, 203
Doob's martingale, 163
Hoeffding's, 80
Jensen's, 54

Kolmogorov's, 163
Markov's, 51
McDiarmid's, 88, 90
inner product, 277
integer program, 239
interactive proof, 263
irreducible, 175
iterated expectation
law of, 41
Iverson bracket, 29
Iverson notation, 29

Jensen's inequality, 54
JLT, 148
Johnson-Lindenstrauss lemma, 142, 147
Johnson-Lindenstrauss transformation, 148
joint distribution, 30
joint probability mass function, 30

Karger's min-cut algorithm, 25
Karp-Upfal-Wigderson bound, 484
ket, 277
key, 102, 114
KNAPSACK, 217
Kolmogorov extension theorem, 17
Kolmogorov's inequality, 163

Las Vegas algorithm, 11
law of iterated expectation, 41
law of total probability, 6, 22
lazy Markov chain, 176
leader election, 14, 292
left spine, 109
lemma
coupling, 174
Johnson-Lindenstrauss, 142, 147
Lovász local, 243
Yao's, 45
length, 361
linear program, 239
linearity of expectation, 7, 32
Lipschitz function, 197
literal, 245
Littlewood-Offord problem, 101
load balancing, 13
load factor, 115
locality-sensitive hashing, 140
locally decodable, 266
locally testable, 266
loop
closed, 125
Lotka-Volterra model, 226
Lovász Local Lemma, 243
Lovász local lemma
sampling, 249
Lyapunov function, 225

magnitude, 279
Markov chain, 165, 166
lazy, 176
Markov chain Monte Carlo, 165
Markov process, 166
Markov's inequality, 51, 52
martingale, 84, 153
Doob, 257
vertex exposure, 91
martingale difference sequence, 84
martingale property, 153
martingale transformation, 135
matching, 209, 299
matrix
stochastic, 276
transition, 167
unitary, 280
max register, 297
maximum cut, 238
McDiarmid's inequality, 88, 90
measurable, 31, 43
measurable function, 28

measurable sets, 16
measure
    probability, 15
measurement, 276
memoryless, 166
message-passing, 289
method of bounded differences, 88
method of conditional probabilities, 257
Metropolis, 182
Metropolis-Hastings
    convergence, 196
Metropolis-Hastings algorithm, 182
Miller-Rabin primality test, 5
min-cut, 25
minimum cut, 25
mixing
    rapid, 185
mixing rate, 201
mixing time, 165, 174
moment, 68
moment generating function, 68
Monte Carlo algorithm, 5, 11
Monte Carlo simulation, 11
multi-armed bandit, 94
multigraph, 25
multiplicative drift, 232

nearest neighbor search, 140
nearly orthogonal, 311
network
    sorting, 235
NNS, 140
no-cloning theorem, 282
non-uniform, 256
norm, 279
normal form
    conjunctive, 239
    disjunctive, 220
notation
    bra-ket, 277
number
    Catalan, 103
number P, 214

objective function, 239
oblivious adversary, 107, 289
open addressing, 115
operator
    Grover diffusion, 286
    Hadamard, 281
optimal solution, 239
optional stopping theorem, 155
orthogonal
    nearly, 311
outcomes, 16

pairwise independence, 62
pairwise independent, 21
paradox
    St. Petersburg, 34
parallel computing, 289
partition, 16
path
    simple, 357
PCP, 262
PCP theorem, 261, 262
PCSA, 135
pending operation, 289
perfect hash function, 121
perfect matching, 211
perfect matchings, 215
period, 176
periodic, 170
permanent, 215
permutation routing, 77
Perron-Frobenius theorem, 199
pessimistic estimator, 258
phase, 281
physical randomness, 9

Physical randomness., 9
pivot, 6
PLEB, 141
point location in equal balls, 141
point query, 137
points, 16
Poisson trial, 487
polynomial-time hierarchy, 215
positive
    false, 128
potential function, 226
prefect hashing
    dynamic, 122
preference, 291
primality test
    Miller-Rabin, 5
priority search tree, 105
Probabilistic Counting with Stochastic Averaging, 135
probabilistic method, 14, 234
probabilistic recurrence, 92
probabilistic recurrence relations, 231
probabilistically-checkable proof, 261, 262
probability, 16
    conditional, 21
    high, 51
    stationary, 171
probability amplification, 253, 255
probability amplitude, 274
probability generating function, 33, 56
probability mass function, 29
probability measure, 15
probability of A conditioned on B, 21
probability of A given B, 21
probability space, 15
    discrete, 16
probability theory, 15
probing, 115
problem
    promise, 273
product
    inner, 277
    tensor, 275
promise problem, 273
proof
    interactive, 263
    probabilistically-checkable, 261
pseudorandom generator, 253
    cryptographically secure, 254
pseudorandom number generator, 10
puzzle
    word search, 442
        futile, 443

quantum bit, 274
quantum circuit, 274, 279
quantum computing, 274
qubit, 274
query
    point, 137
QuickSelect, 49, 483
QuickSort, 6, 92

radix tree, 432
RAM, 1
random circuit, 275
random graph, 29, 91
random structure, 14
random variable, 2, 15, 28
    Bernoulli, 33
    binomial, 33
    discrete, 28, 31
    geometric, 47
    indicator, 29
random walk, 158
    biased, 160
    unbiased, 158
    with one absorbing barrier, 159

with two absorbing barriers, 158
random-access machine, 1
randomized computation, 276
randomized rounding, 220, 240
randomness
physical, 9
range coding, 450
rapid mixing, 185
record, 416
double, 416
red-black tree, 103
reducible, 170
reduction, 214
gap-amplifying, 269
gap-preserving, 269
gap-reducing, 269
register, 289
max, 297
regret, 95
rejection sampling, 182, 216, 450
relax, 239
relaxation, 239
relaxation time, 201
reservoir sampling, 332
reversibility, 171
reversible, 178, 281
right spine, 109
ring-cover tree, 141
rock-paper-scissors, 226
root, 4, 102
rotation, 103
tree, 103
routing
bit-fixing, 78, 208
permutation, 77
run, 361, 455

sampling, 11, 13
rejection, 182, 216, 450
reservoir, 332
sampling Lovász local lemma, 249
SAT, 476
satisfiability, 245
satisfiability problem, 239
satisfy, 239
scapegoat tree, 103
search tree
balanced, 102
binary, 102
priority, 105
second-moment method, 62
seed, 9, 253
separate chaining, 115
set balancing, 235
SHA, 115
shared coin
weak, 292
shared memory, 289
sharp P, 214
sifter, 57, 292, 295
simple path, 357
simplex method, 240
simulated annealing, 196
simulation
Monte Carlo, 11
sketch, 133
count-min, 136
Flajolet-Martin, 134, 433
Tug-of-War, 152
sort
bubble, 364
sorting network, 235
soundness, 262
Space Invaders, 374
spanning tree, 469
spare, 429
spectral Bloom filter, 133
spectral theorem, 199
spine
left, 109

right, 109
sports reporting, 368
sprite, 373
standard deviation, 99
state space, 166
state vector, 274
stationary distribution, 165, 168, 171
stationary probability, 171
Statistical pseudorandomness., 10
stochastic matrix, 167, 276
stochastic process, 166
stopping time, 153, 154
stream
data, 133
strike, 429
strong law of large numbers, 99
strongly k-universal, 116
strongly 2-universal, 115
submartingale, 84, 86, 154
subset sum principle, 265
subtree, 102
supermartingale, 84, 86, 92, 154
symmetry breaking, 14

tabulation hashing, 119
tensor product, 275
termination, 290
test
local, 266
testable
locally, 266
theorem
Adleman's, 255
Berry-Esseen, 100
Doob decomposition, 155
Erdős-Szekeres, 431
Kolmogorov extension, 17
no-cloning, 282
optional stopping, 155
PCP, 261, 262
Perron-Frobenius, 199
spectral, 199
Toda's, 215
Valiant's, 214
third moment, 100
time
coupling, 184, 186
expected, 2
hitting, 224
mixing, 174
relaxation, 201
stopping, 153, 154
time-reversed chain, 180
Toda's theorem, 215
Toffoli gate, 282
total path length, 104
total variation distance, 171
transformation
Johnson-Lindenstrauss, 148
unitary, 280
transition matrix, 167
transition probabilities, 167
transpose
conjugate, 277
treap, 105, 430
tree
AVL, 103
binary, 102
binary search, 102
cartesian, 105
priority search, 105
radix, 432
red-black, 103
ring-cover, 141
rotation, 103
scapegoat, 103
witness, 250
tree rotation, 103
truncation, 156
Tug-of-War sketch, 152

UCB1 algorithm, 95
unary, 146
unbiased random walk, 158
uniform, 256
union bound, 53
unique game, 272
Unique Games Conjecture, 272
unitary matrix, 280
unitary transformation, 280
universal hash family, 115
unranking, 216
upper confidence bound, 95

Valiant's theorem, 214
validity, 290
value, 271
variable drift, 231
variance, 60
vector
    state, 274
vector space, 42
vertex exposure martingale, 91

wait-free, 290
wait-free shared memory, 289
Wald's equation, 36, 162
Walsh-Hadamard code, 265
weak shared coin, 292
with high probability, 51
witness, 5, 256
witness tree, 250
word search puzzle, 442
    futile, 443
worst-case analysis, 2

xxHash, 115

Yao's lemma, 3, 45

Zipf's law, 309
Zobrist hashing, 119